Justification of Matching Outcomes

MSc Thesis *(Afstudeerscriptie)*

written by

Daniela Loustalot Knapp
(born February 27th, 1995 in Ciudad de México, México)

under the supervision of **Prof. Dr. Ulle Endriss**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
January 31st, 2022	Dr. Benno van den Berg *(chair)*
	Arthur Boixel, MSc
	Prof. Dr. Ulle Endriss *(supervisor)*
	Dr. Ronald de Haan

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

This thesis is dedicated to the study of justifications of partial matching outcomes for one-to-one matching problems with two-sided preferences. Matching theory studies the problem of pairing agents from two groups while taking into account their preferences. The idea is to provide non-experts with a step-by-step explanation of why the computed assignment is a good compromise between all the agents' preferences, grounded in a set of principles that are appealing to them. The contribution of this thesis is twofold. Firstly, we define a formal model for justifications. A justification consists of a normative basis together with a step-by-step explanation grounded in the basis and the agents' preferences. Secondly, we show how to automate the search for justifications using SAT solvers. We formally define a procedure to find a justification for a specific problem whenever one exists. We transform the problem of finding a normative basis into a propositional satisfiability problem, so that it can be solved automatically by a SAT solver and we show how to extract an explanation from a minimal unsatisfiable subset. Finally, we present an implementation in Python and some examples of interesting executions.

Acknowledgements

First, I would like to thank Ulle for all his lessons as a teacher and thesis supervisor. By him as a teacher, I was brilliantly introduced to the world of collective decision making, which kept me fascinated and shaped my path during the Master. From him as a supervisor I received all the support. Thanks for all the time invested, the understanding, the honest feedback and encouragement. The process of making this thesis was very enjoyable with such a great supervisor. To the members of my thesis defense committee: Arthur Boixel, Benno van den Berg and Ronald de Haan, thank you for reading my work and making the defense an enriching experience.

Thanks to the matching mechanism that assigned Ronald de Haan as my academic mentor. From day one of the program I felt fully accompanied and his guidance was key during my studies. Thank you Ronald for listening to my ideas and interests and for helping me find my way.

A special thanks goes to Tanja for being so caring and for solving so many of my student problems.

To the friends I made in Amsterdam, thanks for all the shared moments: visiting cafés from a book, going bouldering, eating burgers and facing unexpected situations, enjoying all the music, walking home after the New Year's Eve fireworks, having so many passionate conversations, throwing frisbees all year long, going for workouts in the snow, (hopefully) breaking a world record, enjoying home-made croissants and so much loving. You really made me feel at home. To the friends at the other side of the world, thank you for always staying connected, for keeping me as part of your lives and for being part of mine no matter the distance.

Finally, I would like to thank my family for all the love and support. For being part of every step I take in life, no matter in which direction. To my mom for being an example of strength, courage and love; and for believing in me more than anyone else.

Contents

Listings

1 | Introduction

In this chapter we present the main idea of this work. First, we give some background on matching theory. Then, we motivate the ideas of justifying matching outcomes and of automating the process. Finally, we give an overview of the thesis.

1.1 Matching

A matching problem is the task of pairing agents from two groups while taking into account their preferences. The study of these kinds of problems started with the seminal work of Gale and Shapley (1962), where they proposed the *deferred acceptance algorithm* for what they called the Stable Marriage Problem and showed how to extend it to the College Admission Problem. The Stable Marriage Problem consists of a society with the same number of men and women, each with preferences over those of the opposite sex, where everyone must be married to someone of the other sex.[1] In the College Admission Problem, applicants need to be matched to schools, according to the applicants' preferences and the colleges' preferences and capacities. This problem has been further studied and nowadays is called School Choice (Abdulkadiroğlu and Sönmez, 2003; Abdulkadiroğlu, 2013). Other variations of matching problems can be found in the real world and in the literature. An example is the assignation of residents to hospitals (Roth, 1984). In the US, the National Resident Matching Program[2] matches doctors to hospitals for their residencies. Further applications of matching are the assignment of tenants to houses in a housing allocation situation, the assignation of organ donors to organ receivers (Roth et al., 2004) and the matching of job applicants to vacancies in companies.

Different kinds of matching problems can be classified according to different parameters such as the number of groups of agents involved. Matching problems can consist

[1] The abstract model for this problem is essential in the study of matching problems. For that reason, we may still refer to it but we will instead make the story behind it about a mentorship program where we need to match exactly one student to each mentor.

[2] https://www.nrmp.org/

of two groups of agents (bipartite) or of only one group (non-bipartite). An example of a non-bipartite problem is the Stable Roommates Problem (Manlove, 2013). This problem consists of one group of agents that rank all of the others and it's applied in campus housing allocation, where students have to share a room with one other person (from the same student group). Bipartite problems can be further classified depending on the kind of agents and the type of assignment. We consider economic agents and objects. Economic agents are considered to have preferences whereas objects are not. Thus, we can have problems with one-sided preferences and two-sided preferences. The former means that on one side of the market we have economic agents and on the other we have objects; and the latter means that we are dealing with economic agents on both sides. A matching problem with two-sided preferences is, for example, the mentorship program, whereas a problem with one-sided preferences would be the house allocation problem. Finally, matching problems can be classified according to the type of matching. We can have one-to-one matchings, where each agent from one group is matched to exactly one agent from the other group. The mentorship program and the organ donor problem are examples of one-to-one matching problems. On the other hand, we have one-to-many matchings, where each agent from the left group is assigned to only one element from the right group, but each element from the right group can accept many elements from the left group. Usually in these kinds of problems, the left group is of economic agents and the right group is of objects. Examples of one-to-many matching problems are the hospital/residents problem and the school choice problem. In this work we will be restricted to one-to-one matching problems with two-sided preferences, which correspond to the mentorship program problem.

Different aspects of Matching Theory have been studied throughout the years in various disciplines such as economics, social choice theory and game theory. Some examples are algorithmic aspects like the complexity or feasibility of different mechanisms (Manlove, 2013; Klaus et al., 2016). From an economic and game-theoretic perspective, the strategic behavior of agents has been studied (Roth and Sotomayor, 1992). Moreover, normative aspects have also been studied from an axiomatic perspective (Roth, 1982). We will focus only on the axiomatic study of one-to-one matching problems with two-sided preferences.

1.2 Motivation

Any collective decision making problem can be thought of as a process of aggregating individual interests to arrive at a decision that is a compromise between all the agents'

preferences. As we know from social choice theory, there is no unique good way of doing this and different mechanisms have their advantages and disadvantages, but what is clear is that there is no perfect rule that satisfies every agent's interests. Hence, it is often the case that (some) agents are not happy with the outcome of a collective decision. Thus, it is relevant to focus on the search for a way to extract an explanation of why an outcome of a collective decision is good, for the agents to understand. By a good collective decision we mean one that is aligned with some principles and depends on the concrete instance of the collective decision, i.e., the other agents' preferences. Some work has been done regarding the justification of collective decisions in voting. For instance, Cailloux and Endriss (2016) defined a language to reason about different voting rules with the aim that it could help in automating the analysis of voting rules in a specific situation. Procaccia (2019) argues that axioms should explain the outcomes to the agents involved, in voting. Furthermore, Boixel and Endriss (2020) define a formal model for justifications in voting and they automate the search for justifications using constraint solving. In this work we bring these ideas to the matching setting.

Imagine that we have a set of axioms that we find relevant and (or) acceptable and an instance of a one-to-one matching problem with two-sided preferences, i.e., two groups of agents, each of them with preferences over all the agents from the opposite group. Suppose that we are given a matching that was computed by some mechanism. Suppose that we want to find an explanation for the outcome that is grounded in (some of) the normative principles we selected. But for it to be easily understandable, we would like it to only talk about the specific situation, i.e., we want an explanation that only uses facts of the preference profile that we have. Intuitively, that is what we would want for a justification.

In a matching problem the result of the preference aggregation is local in the sense that it is not the same for all the agents. As a consequence, agents might be interested in an explanation only of the part of the outcome that concerns them. For example, an agent may be wondering why she was matched with her assignee, or why she was not matched with, say, her top preference. In view of this, we will provide a model for the justification of partial outcomes.

As an example, consider that in a university, a mentorship program will be held for the new students at the start of a new academic year. The program consists of assigning a member of the faculty staff as mentor for each student to guide them during their studies. Both the students and the staff members have information on each other and thus, preferences. Both groups submit their preferences and a matching is computed by a central authority in the university. Then everyone is informed of the results. It is very likely that

not everybody will happy with their assignee. It is also likely that not everyone is familiar with the axiomatic method and able to analyze why is it the case that their assignee cannot be better. Furthermore, it is possible that there is a set of principles aligned with the university values, to which both the students and staff members subscribe. In this case, it might be useful to accompany the results of the matching with a personalized explanation, grounded in (some of) these principles, for why each pair is in the outcome. Although, it could be hard in practice to obtain a justification for each pair. In that case, a program that automatically generates justifications, given a set of axioms and the problem instance, would be useful.

In contexts where the axiomatic method is used, SAT solvers have been used for diverse purposes. They have mostly been used to prove impossibility results (Tang and Lin, 2009; Geist and Endriss, 2011) but they have also proved useful in other situations. For instance, to discover theorems in game theory (Tang and Lin, 2011), or for the automated mechanism design for facility location (Okada et al., 2019). Moreover, Endriss (2020) used SAT solvers to prove impossibilities for one-to-one matching problems. Hence, we decided to apply the ideas behind these techniques to automate the search for justifications for partial matching outcomes.

1.3 Thesis Overview

This work is divided as follows. In Chapter 2, we present the basic definitions of one-to-one matching problems, the axioms and some relevant (impossibility) results. Then, in Chapter 3 we define a formal model for a justification of a part of an outcome. For this, we first define a propositional language, the *feature language*, to express features that define partial matchings. Then we prove that the model we defined behaves well with respect to the feature language and we show how to construct justification from others, using the logical structure of the features they justify. We also discuss the property of minimality for normative bases and explanations. Chapter 4 is about the automatic search for justification using SAT solvers. First, we give an overview of the general ideas behind the SAT solving technique in the context of social choice. Then, we present the specific approach we used. We show how to encode a justification problem as a propositional formula; we describe an algorithm that, using this encoding, searches for a justification; and we describe our implementation. Then, we report the results of the program, we present an analysis of its performance and we illustrate how it can be used. Furthermore, we discuss the approach: its advantages and disadvantages. We conclude the thesis in Chapter 5.

2 | Matching Theory

In this chapter we introduce the preliminaries on matching theory. This work is restricted to one-to-one matching problems with two-sided preferences. In the first section (Section 2.1), we introduce the model, which corresponds to the model first defined by Gale and Shapley (1962) for their so-called Stable Marriage Problem. In Section 2.2 we define several normative principles and finally, in Section 2.3 we present some existing (impossibility) results for matching mechanisms.

2.1 The Model

In this section we define a model for one-to-one matching problems with two-sided preferences. In a one-to-one matching problem with two-sided preferences we consider two sets, of size n, of agents. We only consider the most basic variant of this problem and thus we assume that agents have ordinal preferences. That is, they are able to state their preferences in order but they don't necessarily have a quantitative notion on how much they like every agent from the other set. This model corresponds to the Stable Marriage Problem introduced by Gale and Shapley (1962), where in a society with equal number of men and women, everyone needs to get married to someone of the opposite sex. This model can be applied in more relevant situations in real life, for example, in a mentorship program where there are an equal number of junior and senior individuals and we need to assign exactly one mentee to each mentor. Furthermore, it can be applied in a job market where there are the same number of applicants and vacancies.

For the abstraction of these kind of matching problems, we refer to the sets of agents as left agents and right agents, or sometimes in line with the game-theoretic perspective we refer to them as left and right side of the market. We use $\mathcal{L}(X)$ to denote the set of all strict linear orders over a given set X.

The formal model is as follows. An instance of a one-to-one matching problem with two-sided preferences $\mathcal{I}$ is a tuple $\langle n, L, R, \boldsymbol{p}\rangle$. The size or dimension of the problem $\mathcal{I}$ is n. Then we have a set $L = \{\ell_1, \ldots, \ell_n\}$ of n left agents, and a set $R = \{r_1, \ldots, r_n\}$ of

n right agents. Every left agent $\ell \in L$ has a strict linear order $\succ_\ell \in \mathcal{L}(R)$ as preference over the right agents; and every right agent $r \in R$ has a preference order $\succ_r \in \mathcal{L}(L)$ over the left agents. For a given preference order, $\succ$, we use $\succeq$ to denote its reflexive closure, i.e., the corresponding non-strict preference order. The preference orders of all left and right agents constitute a preference profile $\boldsymbol{p} = (\succ_{\ell_1}, \ldots, \succ_{\ell_n}, \succ_{r_1}, \ldots, \succ_{r_n}) \in \mathcal{L}(R)^n \times \mathcal{L}(L)^n$. We use $\mathcal{P}_n$ to denote the set of all profiles for matching problems of size n. That is, $\mathcal{P}_n = \mathcal{L}(R)^n \times \mathcal{L}(L)^n$.

For an agent $a \in L \cup R$, we use $\boldsymbol{p}_{-a}$ to denote the (partial) profile obtained from $\boldsymbol{p}$ by removing agent a's preference order. Moreover, $(\succ'_a, \boldsymbol{p}_{-a})$ denotes the profile obtained from $\boldsymbol{p}$ by replacing $\succ_a$ with $\succ'_a$. If the context is unclear, we use $\succ_a^{\boldsymbol{p}}$ to denote agent's $a \in L \cup R$ preference order in profile $\boldsymbol{p}$. Furthermore, we use $\text{top}_{\boldsymbol{p}}(a)$ to denote agent a's most preferred agent from the other side; and similarly $\text{bot}_{\boldsymbol{p}}(a)$ denotes her least preferred agent.

Then, a matching M is a subset of $L \times R$ such that $|\{r : (\ell, r) \in M\}| = 1$ for every $\ell \in L$ and $|\{\ell : (\ell, r) \in M\}| = 1$ for every $r \in R$. We use $\mathbb{M}_n$ to denote the set of all matchings for a matching problem of size n. When the context is clear we can omit the subscript. Finally, a matching mechanism μ is a function that takes a preference profile and returns a matching:

$$\mu : \mathcal{L}(R)^n \times \mathcal{L}(L)^n \to L \times R$$

For an agent $a \in L \cup R$, we use $\mu(\boldsymbol{p})(a)$ to denote agent a's match computed by mechanism μ under profile $\boldsymbol{p}$.

One of the most relevant examples of a matching mechanisms is the *deferred acceptance algorithm*. This algorithm was the main contribution of the seminal paper of Gale and Shapley (1962), which is considered the origin of matching theory.

Definition 2.1 (Deferred Acceptance Algorithm)**.** The deferred acceptance algorithm for a matching problem $\mathcal{I}$ works in rounds.

(1) In round 1 all left agents *propose* to their most preferred right agent. Every right agent chooses among the received proposals the one she likes the most and rejects all the others.

(n) In round n all left agents that remain unmatched propose to the right agent they like the most and that they haven't yet proposed to. All right agents choose among the received proposals and their temporary match the one they like the most and reject all the others.

The algorithm ends once all agents are matched.

Example 2.1. Consider the following matching problem of size $n = 3$. The preference profile $\boldsymbol{p}$ is represented in the table.

$$\begin{aligned} \ell_1 &: r_1 \succ r_2 \succ r_3 & \qquad r_1 &: \ell_3 \succ \ell_1 \succ \ell_2 \\ \ell_2 &: r_1 \succ r_3 \succ r_2 & \qquad r_2 &: \ell_1 \succ \ell_2 \succ \ell_3 \\ \ell_1 &: r_3 \succ r_2 \succ r_1 & \qquad r_3 &: \ell_2 \succ \ell_3 \succ \ell_1 \end{aligned}$$

Now we illustrate a run of the deferred acceptance algorithm under profile $\boldsymbol{p}$. In the first round, both, agent ℓ_1 and ℓ_2 propose to r_1 and ℓ_3 proposes to r_3; agent r_1 prefers ℓ_1 over ℓ_2, so after the first round the pairs in the (temporary) outcome are (ℓ_1, r_1) and (ℓ_3, r_3). In the second round ℓ_2 sends a proposal to r_3 and as r_3 prefers ℓ_2 over ℓ_3, now the temporary outcome consists of the pairs(ℓ_1, r_1) and (ℓ_2, r_3). In the third round, ℓ_3 proposes to r_2 and r_2 accepts this (her only) proposal. Now the matching is complete and the algorithm terminates. The resulting matching under the deferred acceptance for profile $\boldsymbol{p}$ is:

$$M = \{(\ell_1, r_1), (\ell_2, r_3), (\ell_3, r_2)\}$$

⊣

2.2 Axioms

The theory of matching has been studied from different perspectives. The one we will focus on is the axiomatic method. This method consists of defining normative principles of the objects of study in a formal way and analyzing their consequences. That is, we study when those principles can be satisfied and which ones are compatible. In our case those objects are matching mechanisms. In this section we formally define some of the so-called axioms, or normative principles.

The notion of *stability* was first introduced by Gale and Shapley (1962). The idea behind it is that no pair of agents will have the incentive to act on their own and deviate from the resulting matching to pair up with each other. In this sense, a blocking pair is a pair of agents from opposite sides of the market that prefer each other over their assigned match.

Definition 2.2. A pair of agents $(\ell_i, r_j) \in L \times R$ is a *blocking pair* for matching M and profile $\boldsymbol{p} = (\succ_{\ell_1}, \ldots, \succ_{r_n})$ if there are $\ell_{i'} \in L$ and $r_{j'} \in R$ such that:

- $(\ell_i, r_{j'}) \in M$ and $(\ell_{i'}, r_j) \in M$

- $r_j \succ_{\ell_i} r_{j'}$
- $\ell_i \succ_{r_j} \ell_{i'}$

A matching M is *stable* if there is no blocking pair for M.

Definition 2.3 (Stability)**.** A matching mechanism μ is *stable* if for every profile $\boldsymbol{p}$, the resulting matching, $\mu(\boldsymbol{p})$ is a stable matching.

The deferred acceptance algorithm is an example of a stable mechanism (Gale and Shapley, 1962).

The axiom of stability can be weakened in order to obtain a different stability notion: top-stability, stating that if two agents rank each other in their top preference, then they should be matched to each other.

Definition 2.4 (Top-Stability)**.** A matching mechanism μ is *top-stable* if for every profile $\boldsymbol{p}$, for every left agent $\ell \in L$ and every right agent $r \in R$ if $\text{top}_{\boldsymbol{p}}(\ell) = r$ and $\text{top}_{\boldsymbol{p}}(r) = \ell$ then $(\ell, r) \in \mu(\boldsymbol{p})$.

Then, we have the strategyproofness notions for matching mechanisms. In general, strategyproofness captures the idea that no agent is able to obtain a better outcome by presenting untruthful preferences. As we have two disjoint sets of agents, it makes sense to define one-way strategyproofness notions. First, we need to define when two profiles are a-variant, for some agent $a \in L \cup R$.

Definition 2.5. For an agent $a \in L \cup R$, we say that profiles $\boldsymbol{p}$ and $\boldsymbol{p}$' are *a-variant* if $\boldsymbol{p}_{-a} = \boldsymbol{p}'_{-a}$. That, is, if every agent except (possibly) a has the same preference order in both profiles.

Definition 2.6 (One-Way Strategyproofness)**.** A mechanism μ is *left-strategyproof* if for every left agent $\ell \in L$, for every profile $\boldsymbol{p}$ where $\succ_\ell^{\boldsymbol{p}}$ represents the truthful preferences of ℓ and for every ℓ-variant profile $\boldsymbol{p}'$, it is the case that

$$\mu(\boldsymbol{p}) \succeq_\ell^{\boldsymbol{p}} \mu(\boldsymbol{p}')$$

We define right-strategyproofness in an analogous way.

So, one-way strategyproofness says that every left (right) agent gets a match at least as good when she tells the truth than when she lies. Strategyproofness is obtained in a natural way by putting left and right strategyproofness together.

Definition 2.7 (Strategyproofness)**.** A matching mechanism μ is *strategyproof* if it is left- and right-strategyproof.

Then, we have fairness notions stating the equal treatment of agents. We can think of fairness notions at different levels. For example, both groups of agents being treated equally or agents within a group receiving equal treatment. This notions were defined by Masarani and Gokturk (1989) in the context of the Stable Marriage Problem; this is why they referred to the group-fairness notion as *gender-indifference*.

Definition 2.8 (Peer-indifference)**.** A matching mechanism μ is *peer-indifferent* if for every profile $\boldsymbol{p}$, permutation of the left agents $\pi_L : L \to L$, permutation of the right agents $\pi_R : R \to R$, left agent $\ell \in L$ and right agent $r \in R$, the following holds

$$(\ell, r) \in \mu(\boldsymbol{p}) \text{ if and only if } (\pi_L(\ell), \pi_R(r)) \in \mu((\pi_L \circ \pi_R)(\boldsymbol{p})),$$

where $(\pi_L \circ \pi_R)(\boldsymbol{p})$ is the natural extension of the permutations to the profile.

Endriss (2020) formalized the group-fairness axiom[1] in the context of the model that we are using. First, he defines *swap-variant* profiles. Profiles $\boldsymbol{p}$ and $\boldsymbol{p'}$ are swap-variant if for every $i, j, j' \in \{1, \ldots, n\}$, we have that ℓ_i prefers r_j to $r_{j'}$ in $\boldsymbol{p'}$ whenever r_i prefers ℓ_j to $\ell_{j'}$ in $\boldsymbol{p}$ and the corresponding when replacing left and right agents. Intuitively, swap-variant profiles have the same structure but with the roles of left and right agents swapped.

Definition 2.9 (Group-indifference)**.** A matching mechanism μ is *group-indifferent* if for every pair of swap-variant profiles $\boldsymbol{p}$ and $\boldsymbol{p'}$, and for every $i, j \in \{1, \ldots, n\}$ we have that

$$(\ell_i, r_j) \in \mu(\boldsymbol{p}) \text{ if and only if } (\ell_j, r_i) \in \mu(\boldsymbol{p'}).$$

Next, we have some efficiency axioms. These notions formalize the idea of the resulting matching being *best* (in some sense). But as it is natural, there is no one way of formalizing goodness. So, now we define some properties that define different notions of mechanisms that always return a good, or efficient matching.

The first axiom in this group states that at least one agent is matched to her most preferred option. This principle can also be defined separately for left and right agents.

Definition 2.10 (Top-rewarding)**.** A mechanism μ is *left top-rewarding* if for every profile $\boldsymbol{p}$, there is a left agent $\ell \in L$ such that $(\ell, \text{top}_{\boldsymbol{p}}(\ell)) \in \mu(\boldsymbol{p})$. The definition for *right*

[1] In the paper, the axiom is called *gender-indifference*.

top-rewarding is analogous. We say that μ is *top-rewarding* if it is both, left and right top-rewarding.

The next axiom is inspired in the school choice problem (Abdulkadiroğlu, 2013; Mennle and Seuken, 2014). In the school choice problem it is a desirable property that two students (left agents) are not assigned to schools such that they have an incentive to swap. We call this property in the context of one-to-one matchings with two-sided preferences, swap-stability.

Definition 2.11 (Swap-stability)**.** A matching mechanism μ is *left swap-stable* if for every two left agents $\ell_1, \ell_2 \in L$, it is not the case that $\mu(\boldsymbol{p})(\ell_2) \succ_{\ell_1} \mu(\boldsymbol{p})(\ell_1)$ and $\mu(\boldsymbol{p})(\ell_1) \succ_{\ell_2} \mu(\boldsymbol{p})(\ell_2)$. Right swap-stability is defined in an analogous way and we say that a mechanism μ is *swap-stable* if it is both left and right swap-stable.

Finally, we have a notion dual to top-stability, stating that if two agents rank each other in their bottom preferences, they should not be matched together.

Definition 2.12 (No-Bottoms)**.** A matching mechanism satisfies the axiom *no-bottoms* if for every profile and every pair of agents $\ell \in L$ and $r \in R$, if $\text{bot}_{\boldsymbol{p}}(\ell) = r$ and $\text{bot}_{\boldsymbol{p}}(r) = \ell$ then it should not be the case that $(\ell, r) \in \mu(\boldsymbol{p})$.

These are just some examples of relevant normative principles, but in the literature about matching (Klaus et al., 2016; Manlove, 2013) many more can be found.

2.3 Relevant Results

Typically when using the axiomatic method, we try to study what kinds of properties do different mechanisms satisfy and if sets of axioms are even satisfiable at the same time. In this section we will review some of these results.

A matching mechanism can satisfy more than one axiom, and usually we would like them to satisfy certain sets of them. Unfortunately, not every combination of axioms can be satisfied by a mechanism. This kind of result is called an *impossibility* result. In that sense, if there is a set $\mathcal{A}$ of axioms that cannot be satisfied by a mechanism at the same time, we say that the set is *trivial*, or that the axioms in the set are incompatible. In that sense, a set of axioms is non-trivial if there is at least one mechanism satisfying all of them simultaneously. Some examples of impossibility results are the following.

Proposition 2.1. (Roth, 1982) There is no mechanism for a matching problem size $n \geq 3$ that is stable and strategyproof.

Proposition 2.2. (Endriss, 2020) There is no mechanism for a matching problem size $n \geq 3$ that is top-stable and strategyproof.

Thus, examples of sets of axioms that are trivial are $\{$stability, strategyproofness$\}$ and $\{$top-stability, strategyproofness$\}$, for $n \geq 3$.

2.4 Summary

In this chapter we laid out the basic theory of matching that will be useful for the rest of the work. We defined the formal model of one-to-one matching with two-sided preferences. Furthermore, we gave several examples of normative properties that we will use to base the justifications on. And finally, we defined trivial sets of axioms and presented some examples of impossibilities for the one-to-one matching model.

3 | Justifications

As we already discussed in Section 1.2, the outcome of a matching mechanism is likely to be questioned at a local level, i.e., each agent might wonder why the part of the matching that concerns her (e.g., her assignee) is a good result, or whether it is based on some normative principles that are acceptable to her. Thus, we are interested in justifying a feature that a matching outcome might possess. Such features will be formalizations of statements of the sort: *the pair* (x, y) *belongs to the outcome*, or *agent* x *has to be matched either to* y *or to* z. Given a profile, a justification for a feature will consist of a set of normative principles and an explanation. The normative principles are desirable properties of a matching mechanism, or the so-called axioms in social choice theory, that agents would consider acceptable. The explanation is a set of concrete instances of the normative principles that will constitute a logical argument for why, under the preference profile and once the principles are accepted, the outcome must possess the feature. In this chapter we will present a propositional language to express these kinds of features and a formal model for a justification of a feature of a matching outcome. We will see that the language is consistent with what one would expect from the justifications and we will discuss how to obtain explanations that can arguably be more understandable than others.

Some recent work has been developed concerning justifications in the field of computational social choice for voting. In their paper, Boixel and Endriss (2020) define a model for the justification of a voting outcome based on both normative principles and the specific election profile. In their model, a justification consists of a normative basis and an explanation. In voting, the outcome of an election is one single subset of the alternatives and it is global in the sense that the result is the same for all the agents. In the case of matching, the explanation of a whole outcome may not be so relevant since an agent can only be directly affected by her own assignee or it could even be the case that her match is the only information she has on the whole outcome, in a situation with privacy concerns. Hence, in this work we adapt the model for voting to matching by adding a language that facilitates the expression of local features. The following is an example of what we would want a justification to look like.

Example 3.1. Suppose that in some institution there are three new junior members, so three senior members are selected to mentor them. Both the junior and the senior members have information on each other, so they have preferences over who they would like to work with during the mentorship program. If we take the junior members as left agents and the senior as right agents we can summarize their preferences in the following profile.

$$\begin{aligned} \ell_1 &: r_3 \succ r_1 \succ r_2 & r_1 &: \ell_1 \succ \ell_2 \succ \ell_3 \\ \ell_2 &: r_2 \succ r_1 \succ r_3 & r_2 &: \ell_2 \succ \ell_1 \succ \ell_3 \\ \ell_3 &: r_3 \succ r_2 \succ r_1 & r_3 &: \ell_1 \succ \ell_2 \succ \ell_3 \end{aligned}$$

Suppose that all participants of the mentorship program agree that top-stability (recall Definition 2.4) is a desirable property for the matching process that will be used to pair them up. Furthermore, suppose that some central authority in the institution collects the agents' preferences and computes a matching, claiming that is a good compromise between them. The resulting matching is the following.

$$M = \{(\ell_1, r_3), (\ell_2, r_2), (\ell_3, r_1)\}$$

Observe that agent ℓ_3 is matched with her least preferred right-agent (r_1). So she may question the *fairness* of the process. In this case, we would want to be able to explain to her that $(\ell_3, r_1) \in M$ must be the case given that she agreed with top-stability. We can do that by stating the following. Observe that agents ℓ_1 and r_3 both rank each other as their top preference, so by top-stability, they must be matched to each other. Similarly with agents ℓ_2 and r_2. Thus, the only option left for agent ℓ_3 is to be assigned to r_1. These last sentences will constitute an explanation for the feature of the outcome expressing that the pair (ℓ_3, r_1) has to be in the result under the given profile. The normative basis will consist only of the axiom of top-stability. ⊣

Before presenting the formal model we have to define a language to formally express the features of a matching that we are interested in.

3.1 Feature Language

In order to talk about parts of a matching, we introduce the notion of *feature*. Intuitively, a feature is an expression of some characteristic of a matching that will naturally define a subset of all the possible matchings possessing it. But formally, a feature will be a formula in a propositional language.

Given a dimension n, we define the feature language for problems of size n. This feature language is a propositional language consisting of a set of propositional variables $\text{Var}_n = \{i \rightleftarrows j \,|\, 1 \leq i, j \leq n\}$. The formulas of this language are defined by the following grammar:

$$\varphi ::= i \rightleftarrows j \,|\, \neg\varphi \,|\, \varphi \vee \varphi.$$

For two formulas φ and ψ, we define the formulas $\varphi \wedge \psi$, $\varphi \rightarrow \psi$ and $\varphi \leftrightarrow \psi$ as abbreviations for other formulas in the usual way. For instance, $\varphi \wedge \psi$ stands for $\neg(\neg\varphi \vee \neg\psi)$, and so on.

We define the semantics for the feature language as follows. For any matching $M \in \mathbb{M}_n$, we use $M \models \varphi$ to denote that the formula φ is satisfied by matching M. The satisfaction relation is defined as follows.

$M \models (i \rightleftarrows j)$	if and only if $(\ell_i, r_j) \in M$
$M \models \neg\varphi$	if and only if it is not the case that $M \models \varphi$
$M \models \varphi \vee \psi$	if and only if $M \models \varphi$ or $M \models \psi$

The semantics extend to the rest of the formulas in the natural way. For example, $M \models \varphi \rightarrow \psi$ if and only if $M \models \varphi$ implies $M \models \psi$. For a formula φ, we write $\models \varphi$ to denote that every matching satisfies φ, that is, for every $M \in \mathbb{M}_n$, it holds that $M \models \varphi$. We call such formulas *tautologies*. For example, if the dimension is $n = 3$, then $\models (1 \rightleftarrows 1) \vee (1 \rightleftarrows 2) \vee (1 \rightleftarrows 3)$ holds.

We define the set of models $\text{Mod}(\varphi)$ of a feature formula φ as the set of matchings that satisfy the formula. That is, $\text{Mod}(\varphi) = \{M \in \mathbb{M}_n \,|\, M \models \varphi\}$. The models of a formula φ are exactly the matchings possessing the feature corresponding to the interpretation of φ. So, we say that a matching M has feature φ if $M \in \text{Mod}(\varphi)$, or if equivalently $M \models \varphi$. Moreover, the set of models of a formula φ can be thought of as the set of matchings possessing feature φ.

We will say that two features φ and ψ are *compatible* if the set of matchings possessing both features simultaneously is not empty, i.e., if $\text{Mod}(\varphi) \cap \text{Mod}(\psi) \neq \emptyset$.

Observation 3.1. For feature formulas φ, ψ, we have that

$$\text{Mod}(\varphi) \cap \text{Mod}(\psi) = \text{Mod}(\varphi \wedge \psi)$$

Thus, if two formulas are incompatible it means that there is no model for the conjunction.

Example 3.2. If we consider the feature formula $\varphi = (1 \rightleftarrows 3)$, we have that the matching $M = \{(\ell_1, r_3), (\ell_2, r_2), (\ell_3, r_1)\}$ of Example 3.1 is such that $M \models \varphi$. Furthermore, M satisfies the formula $(1 \rightleftarrows 3) \wedge (2 \rightleftarrows 2) \rightarrow (3 \rightleftarrows 1)$. The set of models of φ is $\text{Mod}(\varphi) = \{M, M'\}$, where $M' = \{(\ell_1, r_3), (\ell_2, r_1), (\ell_3, r_2)\}$. Lastly, we have that φ is compatible with the formula $\psi = (2 \rightleftarrows 2)$ because $\text{Mod}(\varphi \wedge \psi) = \{M\}$ but it is incompatible with $\chi = (1 \rightleftarrows 2)$ because there is no matching where ℓ_1 is paired with r_3 and r_2 at the same time, hence $\text{Mod}(\varphi \wedge \chi) = \emptyset$. ⊣

The purpose of the introduction of the feature language to the model is to express partial features of the outcome. But notice that we are still able to talk about a specific matching. That is, the original idea of a justification can still be realized under this model. Indeed, given a matching $M = \{(\ell_{i_1}, r_{j_1}), \ldots, (\ell_{i_n}, r_{j_n})\}$, we can express in the feature language that a matching is exactly M with the following formula:

$$\bigwedge_{k=1}^{n} (i_k \rightleftarrows j_k)\,.$$

Therefore, the language is expressive enough to talk about both complete and partial matchings.

3.2 The Model

We have defined a language to express features of a matching that can be interpreted as parts of it. So now we are able to express the object we want to justify. In this section we define a formal model for a justification of a feature of a matching outcome.

We define the interpretation $\mathbb{I}(A)$ of an axiom A as the set of matching mechanisms satisfying A. This extends to sets of axioms in the natural way. That is, the set $\mathbb{I}(\mathcal{A})$ is the interpretation of the set of axioms $\mathcal{A}$ if it consists exactly of all mechanisms satisfying every axiom in $\mathcal{A}$. Recall, from Section 2.2 that a set of axioms is *non-trivial* if all of them can be satisfied at the same time by some mechanism. In that sense, whenever a set of axioms $\mathcal{A}$ is non-trivial, we have that $\mathbb{I}(\mathcal{A}) \neq \emptyset$.

Observation 3.2. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are sets of axioms such that $\mathcal{A}_1 \subseteq \mathcal{A}_2$, then $\mathbb{I}(\mathcal{A}_2) \subseteq \mathbb{I}(\mathcal{A}_1)$.This is because if a mechanism satisfies all the axioms in $\mathcal{A}_2$, then it also satisfies all the axioms in $\mathcal{A}_1$.

Before defining the model of justification we need to define the notion of instance of an axiom. In Example 3.1, two different *instances* of the axiom top-stability are used to provide the desired explanation. In the explanation, no reference is made to the full axiom, but only to those parts related both to the specific profile and the pairs that we wanted to make a point about.

The notion of instance makes most sense when we can express the axioms in a formal language. Moreover, a precise definition is dependent on the language. In their paper, Boixel and Endriss (2020), provide a more general description of what an instance is. For now that will be enough for us to work with. The conditions are the following.

(i) If A' is an instance of an axiom A, then A' must itself be an axiom.

(ii) The interpretation of an axiom is equal to the intersection of the interpretations of its instances. This implies that if A' is an instance of A, then $\mathbb{I}(A) \subseteq \mathbb{I}(A')$.

(iii) The number of instances of an axiom is finite.

We use $A' \triangleleft A$ to denote that A' is an instance of A. Similarly, if $\mathcal{A}'$ and $\mathcal{A}$ are sets of axioms, we use $\mathcal{A}' \triangleleft \mathcal{A}$ to denote that every axiom of $\mathcal{A}'$ is an instance of some axiom in $\mathcal{A}$.

Observation 3.3. The conditions of being an instance imply that if $\mathcal{A}$ and $\mathcal{A}'$ are sets of axioms such that $\mathcal{A}' \triangleleft \mathcal{A}$, then $\mathbb{I}(\mathcal{A}) \subseteq \mathbb{I}(\mathcal{A}')$.

We have defined all the notions that we will use for the justification model. The situation where we will be interested in obtaining a justification consists of the following. First, a matching problem, defined by a preference profile, which is the specific situation the agents find themselves in. Then, the set of relevant normative principles for the agent(s), which we will call a corpus. And finally, a formula in the feature language, the interpretation of which will be the feature of the outcome that we are interested in justifying. These elements will constitute a *justification problem*.

Definition 3.1 (Justification Problem)**.** A *justification problem* is a tuple $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$, where $\mathbb{A}$ is a corpus of axioms for matching mechanisms; $\boldsymbol{p}$ is a preference profile for the one-to-one matching problem in question; and φ is a feature of matchings, or formally, a formula in the feature language.

Given a justification problem, we formally define a justification for the resulting matching possessing the feature in question. Intuitively, a justification will make it possible to understand why, given some principles and a certain situation, the outcome will necessarily possess the relevant feature.

Definition 3.2 (Justification)**.** Given a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$, we say that a pair of sets of axioms $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a *justification for the resulting matching under* $\boldsymbol{p}$ *having feature* $\boldsymbol{\varphi}$ if the following conditions hold.

(i) **Explanatoriness.** $\mathcal{A}^E$ can explain the resulting matching having feature φ:

$$\mu(\boldsymbol{p}) \models \varphi, \text{ or equivalently } \mu(\boldsymbol{p}) \in \text{Mod}(\varphi), \text{ for every mechanism } \mu \in \mathbb{I}(\mathcal{A}^E).$$

(ii) **Relevance.** The explanation $\mathcal{A}^E$ is an instance of the normative basis $\mathcal{A}^N$:

$$\mathcal{A}^E \triangleleft \mathcal{A}^N.$$

(iii) **Adequacy.** All the axioms in the normative basis belong to the corpus of axioms provided:

$$\mathcal{A}^N \subseteq \mathbb{A}$$

(iv) **Non-triviality.** The normative basis is non-trivial:

$$\mathbb{I}(\mathcal{A}^N) \neq \emptyset.$$

Notice that in the definition, φ is a formula in the language, so strictly speaking the justification is for the resulting matching having the feature that corresponds to the interpretation of the feature formula. Furthermore, unlike Boixel and Endriss (2020) in their definition of a justification for a voting outcome, we don't require the explanation to be minimal. That is, such that no proper subset of it is an explanation too. Moreover, we don't require the normative basis to be minimal, this is aligned with the definition for justifications for voting outcomes.

In Section 3.4 we will discuss both design choices: not requiring neither the normative basis nor the explanation to be minimal. We will explain why we decide to leave them as properties of a justification instead of part of the definition. Furthermore, in the case of explanations, we give a procedure to obtain a minimal explanation from an explanation that need not be minimal, showing that not requiring minimality doesn't make our definition less precise.

Example 3.3. Consider the following profile $\boldsymbol{p}$:

$$\ell_1 : r_1 \succ r_2 \succ r_3 \qquad r_1 : \ell_3 \succ \ell_1 \succ \ell_2$$

$$\ell_2 : r_1 \succ r_3 \succ r_2 \qquad r_2 : \ell_1 \succ \ell_2 \succ \ell_3$$
$$\ell_1 : r_3 \succ r_2 \succ r_1 \qquad r_3 : \ell_2 \succ \ell_3 \succ \ell_1$$

Suppose that agent ℓ_1 has a very strong preference for her first option (agent r_3) over the rest, while she agrees that the principles of top-stability (TOPSTA) and left-strategyproofness (LSP) are desirable properties for a matching mechanism. Then, we would want to justify the fact that in every resulting matching computed by a mechanism that satisfies at least those principles, agents ℓ_1 and r_3 cannot be paired together.

For, take $\mathbb{A}$ as any corpus of axioms that contains top-stability (Definition 2.4) and left strategyproofness (Definition 2.6). Formally, take $\mathbb{A}$ as any corpus of axioms such that $\{\mathsf{TOPSTA}, \mathsf{LSP}\} \subseteq \mathbb{A}$.

Then, the pair $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$, is a justification for the feature formula $\varphi = \neg(1 \rightleftarrows 3)$ where $\mathcal{A}^N = \{\mathsf{TOPSTA}, \mathsf{LSP}\}$ and the explanation $\mathcal{A}^E$ is as follows. Consider the next profile $\boldsymbol{p}'$ where every agent but ℓ_1 has the same preferences.

$$\ell_1 : r_2 \succ r_1 \succ r_3 \qquad r_1 : \ell_3 \succ \ell_1 \succ \ell_2$$
$$\ell_2 : r_1 \succ r_3 \succ r_2 \qquad r_2 : \ell_1 \succ \ell_2 \succ \ell_3$$
$$\ell_3 : r_3 \succ r_2 \succ r_1 \qquad r_3 : \ell_2 \succ \ell_3 \succ \ell_1$$

Then, the following is an instance of the axiom LSP when we fix profile $\boldsymbol{p}$ as the original profile, agent ℓ_1 as the manipulator and $\succ'_{\ell_1} = r_2 \succ r_1 \succ r_3$ as her untruthful preference. Notice that profile $\boldsymbol{p}'$ is exactly the resulting profile of only agent ℓ_1 reporting untruthful preferences.

$$\mathsf{LSP}_1 : \quad \text{If } \mu(\boldsymbol{p})(\ell_1) = r_3, \text{ then } \mu(\boldsymbol{p}')(\ell_1) \neq r_2$$

The following is an instance when fixing profile $\boldsymbol{p}'$ of TOPSTA.

$$\mathsf{TOPSTA}_1 : \quad \text{If } \text{top}_{\boldsymbol{p}'}(\ell_1) = r_2 \text{ and } \text{top}_{\boldsymbol{p}'}(r_2) = \ell_1, \text{ then } (\ell_1, r_2) \in \mu(\boldsymbol{p}')$$

Then, the explanation consists of these instances of TOPSTA and LSP. That is, $\mathcal{A}^E = \{\mathsf{TOPSTA}_1, \mathsf{LSP}_1\}$. One can see that this set can indeed explain that the outcome has to have feature φ. Suppose for the sake of a contradiction that the outcome need not possess feature φ. That is, suppose that (ℓ_1, r_3) can be part of the resulting matching. Observe that in profile $\boldsymbol{p}'$, agents ℓ_1 and r_2 have to be matched, by top-stability because they are ranking each other in their top preferences. But if this is the case and , then

in profile $\boldsymbol{p}$, agent ℓ_1 could manipulate to be matched to r_2 rather than to r_3, which she prefers. Hence, the resulting matching from a mechanism that is at least top-stable and left strategyproof has to have feature φ. This proves the explanatoriness condition of the justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$. Relevance and adequacy hold because of the way we chose $\mathcal{A}^N$ and $\mathcal{A}^E$. Non-triviality holds because there is at least one mechanism that satisfies both top-stability and left-strategyproof. Thus, indeed $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a justification for the problem. ⊣

Justifications are not unique. We can find justifications for the same problem based on different sets of normative principles. It can even be the case that a problem is justified under normative bases conformed by axioms that are not compatible. This opens the possibilities for offering completely different explanations. Such a thing could be useful when agents have different views on which properties they value. Even if they find different axioms appealing, it would still be possible to provide them with a justification. For instance, the problem from the previous example can also be justified with a different normative basis.

Example 3.4. Take the justification problem from the previous example (Example 3.3). That is, $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ where $\boldsymbol{p}$ is the profile depicted there, $\varphi = \neg (1 \rightleftarrows 3)$ and the corpus $\mathbb{A}$ is such that $\mathbb{A} \subseteq \{\mathsf{TOPSTA}, \mathsf{LSP}\}$. Now suppose that the axiom no-bottoms (NBOT) is the one contained in the corpus.

Thus, if we consider the normative basis $\mathcal{A}_2^N = \{\mathsf{NBOT}\}$ and the explanation $\mathcal{A}_2^E = \{\mathsf{NBOT}_1\}$, where

$$\mathsf{NBOT}_1: \quad \text{If } \mathrm{bot}_{\boldsymbol{p}}(\ell_1) = r_3 \text{ and } \mathrm{bot}_{\boldsymbol{p}}(r_3) = \ell_1 \text{ then } (\ell_1, r_3) \notin \mu(\boldsymbol{p}),$$

then the pair $\langle \mathcal{A}_2^N, \mathcal{A}_2^E \rangle$ is a justification for the same problem. Observe that this time the outcome not containing pair (ℓ_1, r_3) can be justified by the principle stating that if two agents rank each other in their bottom preference they should not be matched together, and the fact that agents ℓ_1 and r_3 rank each other in their bottom preference. ⊣

So, we know that it is possible to find several justifications for the same feature. Now the opposite question arises, whether it is possible to justify two different features, in the same situation, with the same normative basis. That would be undesirable in the case where the features express conflicting characteristics of a matching. We prove that this is not the case. Moreover, we prove that if the features are incompatible, they cannot be justified with normative bases contained in each other. A similar result in the model for voting is proven by Boixel and Endriss (2020). Theorem 1 in their paper states that it is

impossible to give justifications grounded in the same normative basis for two different election outcomes.

Proposition 3.1. It is not possible to have justifications for incompatible features under the same profile grounded in normative bases $\mathcal{A}_1^N$ and $\mathcal{A}_2^N$ such that $\mathcal{A}_1^N \subseteq \mathcal{A}_2^N$.

Proof. For the sake of a contradiction, suppose that there are justifications $\langle \mathcal{A}_1^N, \mathcal{A}_1^E \rangle$ and $\langle \mathcal{A}_2^N, \mathcal{A}_2^E \rangle$ for some problems $\langle \mathbb{A}, \boldsymbol{p}, \varphi_1 \rangle$ and $\langle \mathbb{A}, \boldsymbol{p}, \varphi_2 \rangle$, respectively, such that φ_1 and φ_2 are incompatible. That is $\text{Mod}(\varphi_1) \cap \text{Mod}(\varphi_2) = \emptyset$.

We will first show that $\mathbb{I}(\mathcal{A}_1^E) \cap \mathbb{I}(\mathcal{A}_2^E) = \emptyset$. Proceeding by contradiction, suppose that $\mu \in \mathbb{I}(\mathcal{A}_1^E) \cap \mathbb{I}(\mathcal{A}_2^E)$. Then $\mu \in \mathbb{I}(\mathcal{A}_1^E)$. By explanatoriness, $\mu(\boldsymbol{p}) \in \text{Mod}(\varphi_1)$. Similarly, $\mu \in \mathbb{I}(\mathcal{A}_2^E)$ and thus $\mu \in \text{Mod}(\varphi_2)$. Hence, $\mu \in \text{Mod}(\varphi_1) \cap \text{Mod}(\varphi_2)$. But we assumed that φ_1 and φ_2 are incompatible, i.e., that $\text{Mod}(\varphi_1) \cap \text{Mod}(\varphi_2) = \emptyset$. Therefore, there cannot be such $\mu \in \mathbb{I}(\mathcal{A}_1^E) \cap \mathbb{I}(\mathcal{A}_2^E)$ and we conclude that $\mathbb{I}(\mathcal{A}_1^E) \cap \mathbb{I}(\mathcal{A}_2^E) = \emptyset$.

Now by Observation 3.2 we have that $\mathcal{A}_1^N \subseteq \mathcal{A}_2^N$ implies that $\mathbb{I}(\mathcal{A}_2^N) \subseteq \mathbb{I}(\mathcal{A}_1^N)$. Then, by relevance we have that both $\mathcal{A}_1^E \triangleleft \mathcal{A}_1^N$ and $\mathcal{A}_2^E \triangleleft \mathcal{A}_2^N$. This, by Observation 3.3 implies that $\mathbb{I}(\mathcal{A}_1^N) \subseteq \mathbb{I}(\mathcal{A}_1^E)$ and $\mathbb{I}(\mathcal{A}_2^N) \subseteq \mathbb{I}(\mathcal{A}_2^E)$. Hence, $\mathbb{I}(\mathcal{A}_2^N) \subseteq \mathbb{I}(\mathcal{A}_1^E) \cap \mathbb{I}(\mathcal{A}_2^E)$.

But we had proven that $\mathbb{I}(\mathcal{A}_1^E) \cap \mathbb{I}(\mathcal{A}_2^E) = \emptyset$, so it must be the case that $\mathbb{I}(\mathcal{A}_2^N) = \emptyset$. Which is in contradiction with the non-triviality condition for the justification for φ_2. Hence, such justifications cannot exist. □

We have defined the model for a justification for a feature, or a partial matching outcome. In the following sections we will discuss how our definition of a justification behaves in comparison with the structure of the feature language (Section 3.3). Furthermore, we will discuss the minimality properties for normative basis as well as for explanations (Section 3.4).

3.3 Compatibility with the Logic

In this section we analyze the structure of the collection of all justifications, we show how this structure is compatible with the feature language and that then we can obtain justifications for certain features by the means of the logical structure of the feature. We show how to obtain such justifications. Furthermore, we show what do some justifications look like depending on the logical structure of the feature formula.

We start with the limit cases. We see what a justification looks like for trivial formulas such as contradictions and tautologies. First we show that, as expected, there is no justification for a contradiction. A contradiction is a formula that has no models. For example, when $n = 3$, the feature formula $\varphi = (1 \rightleftarrows 1) \wedge (1 \rightleftarrows 2)$ is a contradiction because, by

definition, a matching is such that every left agent is matched to at most (and exactly) one right agent. So no matching would be a model for φ.

Proposition 3.2. Let $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ be a justification problem such that φ is a contradiction. That is, $\text{Mod}(\varphi) = \emptyset$. Then no pair $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ forms a justification for φ.

Proof. Notice that by non-triviality ($\mathbb{I}(\mathcal{A}^N) \neq \emptyset$) and relevance ($\mathcal{A}^E \triangleleft \mathcal{A}^N$), the set $\mathbb{I}(\mathcal{A}^E)$ is non-empty. Then, for every $\mu \in \mathbb{I}(\mathcal{A}^E)$ we have that $\mu \notin \text{Mod}(\varphi)$ because $\text{Mod}(\varphi) = \emptyset$. Hence, every choice of $\mathcal{A}^N$ and $\mathcal{A}^E$ cannot satisfy non-triviality, relevance and explanatoriness at the same time and thus it cannot form an explanation for φ. □

Proposition 3.2 is a nice soundness property of the justification model, it entails that we can only justify features that at least some matching has.

On the contrary, every non-trivial set of axioms can justify a tautology. An example of a tautology for a matching problem of dimension n, as mentioned in Section 3.1, is the formula $(1 \rightleftarrows 1) \vee \cdots \vee (1 \rightleftarrows n)$, interpretation of which is that left agent ℓ_1 is matched to at least one right agent.

Proposition 3.3. Let $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ be a justification problem such that φ is a tautology. That is, $\text{Mod}(\varphi)$ is the set of all matchings. Then for any non-trivial set $\mathcal{A}^N \subseteq \mathbb{A}$ and every set $\mathcal{A}^E$ of instances of $\mathcal{A}^N$, the pair $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a justification for φ.

Proof. Every mechanism $\mu \in \mathbb{I}(\mathcal{A}^E)$ is such that $\mu(\boldsymbol{p}) \in \text{Mod}(\varphi)$ because φ is a tautology. Hence, the explanatoriness condition holds. Relevance, adequacy and non-triviality hold by assumption. Therefore, $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a justification for φ. □

Proposition 3.3 implies that even the empty set can explain a tautology. This is apparently a trivial property but it is nice to have. It can be interpreted as the fact that a feature that every matching has, needs no further explanation. This implies that a minimal explanation will only explain meaningful (non-trivial) features.

Now, we show how to obtain justifications for features from those for formulas that are logically related. In a logical language we construct formulas from others by means of logical connectives. Thus, we consider how a justification for a new formula can be constructed from those for the original feature formulas, and if it is at all feasible. We will consider feature formulas with different structures.

First we consider implication. A feature formula ψ is a logical consequence of a formula φ, or φ logically entails ψ (in symbols $\models \varphi \to \psi$) if for every matching M, it is the case that if $M \models \varphi$, then $M \models \psi$. An example of a feature ψ that is a logical consequence of another feature φ, when $n = 3$, is $\psi = (1 \rightleftarrows 2) \vee (1 \rightleftarrows 3)$, if $\varphi = \neg(1 \rightleftarrows 1)$. A justification for a formula also justifies all its logical consequences.

Proposition 3.4. If there are two feature formulas φ and ψ, such that $\models \varphi \to \psi$ and there is a justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ for $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$, then $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ also is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \psi \rangle$.

Proof. We first prove that explanatoriness is satisfied. Let $\mu \in \mathbb{I}(\mathcal{A}^E)$. By explanatoriness of $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ as a justification for φ, we know that $\mu(\boldsymbol{p}) \models \varphi$. Then, as $\models \varphi \to \psi$, it follows that $\mu(\boldsymbol{p}) \models \psi$.

Relevance, adequacy and non-triviality follow directly. Hence, $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \psi \rangle$. □

Two formulas φ and ψ are logically equivalent if they are a logical consequence of each other, i.e., if $\models \varphi \to \psi$ and $\models \psi \to \varphi$. We denote logical equivalence by $\models \varphi \leftrightarrow \psi$. Examples of logical equivalent features when the matching problem is of size 2, are $\varphi = \neg(1 \rightleftarrows 1)$ and $\psi = (1 \rightleftarrows 2)$. As a consequence of Proposition 3.4, a justification for a feature formula justifies any logically equivalent feature.

Corollary 3.1. *If there is a justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ for $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ and ψ is a feature formula equivalent to φ i.e., such that $\models \varphi \leftrightarrow \psi$, then $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \psi \rangle$.*

Then, we examine the conditions to construct a justification for the conjunction of feature formulas. The conjunction of feature formulas is an interesting case because joining two features gives more information about the matching outcome. Thus, if we have justifications for two (compatible) features, we may be interested in obtaining a justification for the conjunction. We show how to obtain such justifications, whenever it is possible.

First we consider the case where the justifications of both conjuncts are based on the same normative principles. In this case we can just *concatenate* the explanations to obtain a justification for the conjunction (of the features).

Proposition 3.5. Let φ_1 and φ_2 be features of matchings. Let $\mathbb{A}$ be a corpus of axioms and let $\boldsymbol{p}$ be a preference profile for a matching problem $\mathcal{I}$. Suppose $\langle \mathcal{A}^N, \mathcal{A}_1^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi_1 \rangle$ and $\langle \mathcal{A}^N, \mathcal{A}_2^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi_2 \rangle$. That is, there are justifications for φ_1 and φ_2 grounded in the same basis. Then $\langle \mathcal{A}^N, \mathcal{A}_1^E \cup \mathcal{A}_2^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi_1 \wedge \varphi_2 \rangle$.

Proof. Let's prove the conditions for a justification one by one.

(i) Explanatoriness. Let $\mu \in \mathbb{I}(\mathcal{A}_1^E \cup \mathcal{A}_2^E)$. By Observation 3.2 and because $\mathcal{A}_1^E \subseteq \mathcal{A}_1^E \cup \mathcal{A}_2^E$ and $\mathcal{A}_2^E \subseteq \mathcal{A}_1^E \cup \mathcal{A}_2^E$, we have that $\mu \in \mathbb{I}(\mathcal{A}_1^E)$ and $\mu \in \mathbb{I}(\mathcal{A}_2^E)$. By the explanatoriness condition on the original justifications, we have that $\mu(\boldsymbol{p}) \in \text{Mod}(\varphi_1)$ and $\mu(\boldsymbol{p}) \in \text{Mod}(\varphi_1)$. Hence, $\mu(\boldsymbol{p}) \in \text{Mod}(\varphi_1) \cap \text{Mod}(\varphi_2)$ and by Observation 3.1, it holds that $\mu(\boldsymbol{p}) \in \text{Mod}(\varphi_1 \wedge \varphi_2)$.

(ii) Relevance. As both $\mathcal{A}_1^E$ and $\mathcal{A}_2^E$ are instances of the normative basis $\mathcal{A}^N$, so is their union.

(iii) and (iv) Adequacy and Non-triviality are preserved by the normative basis. □

Observe that obtaining a justification for a conjunction in this situation is always possible. That is, if there are justifications for features φ and ψ and the same profile, respectively, grounded in the same basis, then the features must be compatible and we can provide a justification for the conjunction. This follows from Proposition 3.1. But more generally, we can always justify a conjunction of compatible features by putting the two justifications together as long as the union of the normative bases is non-trivial (Proposition 3.6).

Proposition 3.6. Let φ_1 and φ_2 be compatible features of matchings. Let $\mathbb{A}$ be a corpus of axioms and $\boldsymbol{p}$ be a preference profile for a matching problem $\mathcal{I}$. Suppose that $\langle \mathcal{A}_1^N, \mathcal{A}_1^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi_1 \rangle$, that $\langle \mathcal{A}_2^N, \mathcal{A}_2^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi_2 \rangle$ and that $\mathcal{A}_1^N \cup \mathcal{A}_2^N$ is a non-trivial set of axioms, i.e., that $\mathbb{I}(\mathcal{A}_1^N \cup \mathcal{A}_2^N) \neq \emptyset$. Then $\langle \mathcal{A}_1^N \cup \mathcal{A}_2^N, \mathcal{A}_1^E \cup \mathcal{A}_2^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi_1 \wedge \varphi_2 \rangle$.

Proof. We need to prove the conditions of the definition of justification. In this case we prove relevance first, since we use it to prove explanatoriness.

(ii) *Relevance.* Let $A \in \mathcal{A}_1^E \cup \mathcal{A}_2^E$. If $A \in \mathcal{A}_1^E$, then by relevance of the justification for φ_1, there is a $B \in \mathcal{A}_1^N$ such that $A \triangleleft B$. Analogously if $A \in \mathcal{A}_2^E$. So, for every $A \in \mathcal{A}_1^E \cup \mathcal{A}_2^E$, there is a $B \in \mathcal{A}_1^N \cup \mathcal{A}_2^N$ such that $A \triangleleft B$. Thus, $\mathcal{A}_1^E \cup \mathcal{A}_2^E \triangleleft \mathcal{A}_1^N \cup \mathcal{A}_2^N$.

(i) *Explanatoriness.* First, we have that $\mathbb{I}(\mathcal{A}_1^E \cup \mathcal{A}_2^E) \neq \emptyset$ because by assumption, $\mathbb{I}(\mathcal{A}_1^N \cup \mathcal{A}_2^N) \neq \emptyset$ and by relevance, $\mathbb{I}(\mathcal{A}_1^N \cup \mathcal{A}_2^N) \subseteq \mathbb{I}(\mathcal{A}_1^E \cup \mathcal{A}_2^E)$. So, let $\mu \in \mathbb{I}(\mathcal{A}_1^E \cup \mathcal{A}_2^E)$. Then, $\mu \in \mathbb{I}(\mathcal{A}_1^E)$ because $\mathbb{I}(\mathcal{A}_1^E \cup \mathcal{A}_2^E) \subseteq \mathbb{I}(\mathcal{A}_1^E)$. By explanatoriness of the first justification, we have that $\mu(\boldsymbol{p}) \models \varphi_1$. By a similar argument it follows that $\mu(\boldsymbol{p}) \models \varphi_2$. Hence, $\mu(\boldsymbol{p}) \models \varphi_1 \wedge \varphi_2$.

(iii) *Adequacy.* We have that $\mathcal{A}_1^N \subseteq \mathbb{A}$ and $\mathcal{A}_2^N \subseteq \mathbb{A}$. Thus, $\mathcal{A}_1^N \cup \mathcal{A}_2^N \subseteq \mathbb{A}$.

(iv) *Non-triviality.* Holds by assumption. □

We turn our attention to the disjunction of feature formulas. The first thing we can observe is that if there is a justification for a feature formula that is a disjunction, say $\varphi \vee \psi$ and there is a justification for the negation of one of the disjuncts, say for $\neg\varphi$, we can put them together to obtain a justification of ψ, as long as the disjunction and the negation are compatible and the normative bases are compatible too.

Observation 3.4. Assume that $\langle \mathcal{A}_1^N, \mathcal{A}_1^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi \vee \psi \rangle$ and that $\langle \mathcal{A}_2^N, \mathcal{A}_2^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \neg\varphi \rangle$. If there is a justification for $(\varphi \vee \psi) \wedge \neg\varphi$, then it is a justification for ψ.

This is a consequence of both Corollary 3.1 and Proposition 3.6. Notice that it is nevertheless not the case that if there is a justification for a disjunction $\varphi \vee \psi$, there need be a justification either for φ or for ψ.

Example 3.5. Recall Example 3.4 but consider the corpus of axioms being exactly the set $\mathbb{A} = \{\mathsf{NBOT}\}$. Observe that this doesn't change the argument of $\langle \mathcal{A}^N, \mathcal{A}^E \rangle = \langle \{\mathsf{NBOT}\}, \{\mathsf{NBOT}_1\} \rangle$ being a justification for the formula $\varphi = \neg\,(1 \rightleftarrows 3)$.

Now consider the formula $\psi = (1 \rightleftarrows 1) \vee (1 \rightleftarrows 2)$. Observe that it is equivalent to φ. That is, $\models \varphi \leftrightarrow \psi$. Hence, by Corollary 3.1 $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a justification for ψ.

It is then easy to see that, with respect to the corpus $\mathbb{A}$, there is no way of justifying either the feature formula $(1 \rightleftarrows 1)$ nor the formula $(1 \rightleftarrows 2)$. Thus, this is a situation where there exists a justification for a disjunction but there is no justification for either of the disjuncts. ⊣

We have seen when it is possible and how to obtain justifications for feature by only looking at their logical structure. Furthermore, we saw how the model of a justification that we defined in Section 3.2 behaves as one would expect in relation to the structure of the feature formulas. That is, for example, that we are not able to justify contradictions and that tautologies are trivially justifiable. This results will become important when we automate the process for the search of justifications. In some cases, it will be easier to for example, search for a justification for a feature that is equivalent to the one that we originally wanted to justify.

3.4 Minimal Justifications

When defining a justification (Definition 3.2), we didn't require either the normative basis nor the explanation to be minimal. By minimal we mean that no proper subset is a normative basis or an explanation, respectively. These were design choices. Not requiring minimality for the basis is aligned with the definition of Boixel and Endriss (2020) of a justification for voting outcomes, while our choice of not requiring minimality for the explanation is not. In this section we discuss why we decided not to force minimality for neither of the parts of a justification. Furthermore, in the case of explanations, we present a procedure to obtain a minimal explanation from one that might not be so.

First we formally define minimality for normative bases and explanations.

Definition 3.3 (Minimal Normative Basis)**.** Given a justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ for a problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$, we say that the normative basis $\mathcal{A}^N$ is minimal if there is no proper subset $\mathcal{A} \subset \mathcal{A}^N$ such that there exists a set of instances $\mathcal{A}' \triangleleft \mathcal{A}$ such that the pair $\langle \mathcal{A}, \mathcal{A}' \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$.

Definition 3.4 (Minimal Explanation)**.** Given a justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ for a problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$, we say that the explanation $\mathcal{A}^E$ is minimal if for every proper subset $\mathcal{A} \subset \mathcal{A}^E$, $\mu(\boldsymbol{p}) \not\models \varphi$ for some $\mu \in \mathbb{I}(\mathcal{A})$.

As we consider minimal normative bases and minimal explanations, we say that a justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ for a problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ is *minimal* if both, $\mathcal{A}^N$ and $\mathcal{A}^E$ are minimal.

3.4.1 Minimal Normative Bases

A smaller normative basis can be desirable because we may be interested in having a weaker set of axioms the justifications are grounded in. It may also be easier for the agents to keep in mind, or to understand a smaller number of normative principles. Nevertheless, we decided not to require minimality for normative bases mainly for two reasons. Firstly, as it is already known, many axioms are in conflict with each other, so it is unlikely that in practice a basis of big size would be found. Secondly, in practice it can be the case that an explanation is *shorter* if it involves more axioms, so if that's the case, we may prefer the justification with the bigger normative basis. We illustrate this with the next example where the explanation for a feature becomes shorter and easier to understand if we add one more axiom.

Example 3.6. Consider a matching problem of size 3 where the preference profile $\boldsymbol{p}$ is as follows.

$$\begin{aligned} \ell_1 &: r_2 \succ r_1 \succ r_3 \qquad & r_1 &: \ell_1 \succ \ell_2 \succ \ell_3 \\ \ell_2 &: r_1 \succ r_2 \succ r_3 \qquad & r_2 &: \ell_2 \succ \ell_3 \succ \ell_1 \\ \ell_3 &: r_3 \succ r_2 \succ r_1 \qquad & r_3 &: \ell_3 \succ \ell_2 \succ \ell_1 \end{aligned}$$

Let the feature formula be $\varphi = \neg(1 \rightleftarrows 1)$ and let the corpus $\mathbb{A}$ be such that it contains the axioms of left swap-stability (LSS, Definition 2.11) and top-stability (Definition 2.4). That is, $\{\mathsf{LSS}, \mathsf{TOPSTA}\} \subseteq \mathbb{A}$. Consider the justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$.

First, we show that there is a justification for φ with normative basis $\{\mathsf{LSS}\}$. Observe that in profile $\boldsymbol{p}$, left agent ℓ_1 prefers r_2 over r_1 and ℓ_2 prefers agent r_1 over r_2. So take the instance, say LSS_1, of left swap-stability, saying that either the pair (ℓ_1, r_1) or the pair

(ℓ_2, r_2) is not in the matching computed under profile $\boldsymbol{p}$.[1] Let LSS_2 be the instance of left swap-stability saying that under profile either the pair (ℓ_3, r_2) or the pair (ℓ_2, r_3) is not in the resulting matching. Now, the explanation for this feature is the set $\{\mathsf{LSS}_1, \mathsf{LSS}_2\}$. And the human-readable version of it would look like the following argument.

Recall that we want to explain the fact that the pair (ℓ_1, r_1) is not in the outcome. We start from LSS_1. As it is a disjunction, we will make a case distinction. One of the disjuncts is the feature that we want to prove, so if that's the case we are done. For the other case, suppose that the pair (ℓ_2, r_2) is not in the matching outcome under profile $\boldsymbol{p}$. As the resulting matching is well defined, agent ℓ_2 has to be matched to some right agent. There are two options, either she is assigned to r_1 or to r_3. Assume the former, then agent ℓ_1 cannot be matched to r_1 (because ℓ_2 already is), so we have proven that the pair (ℓ_1, r_1) is not in the resulting matching. Finally, if ℓ_2 is matched to r_3, by LSS_2, it should be the case that the pair (ℓ_3, r_2) is not in the resulting matching. But as ℓ_3 is matched to r_2, then also the pair (ℓ_3, r_3) is not in the outcome. Then, as ℓ_3 has to be matched to some right agent, it should be that the pair (ℓ_3, r_1) is in the outcome. Therefore, ℓ_1 cannot be assigned r_1. We have then proven that in any case, the pair (ℓ_1, r_1) is not in the outcome.

Observe that the argument, while logically valid is quite confusing. Moreover, using only the axiom left swap-stability, there is no much better way to provide an explanation for the feature. Now consider the normative basis $\{\mathsf{LSS}, \mathsf{TOPSTA}\}$. That is, now we add the axiom of top-stability to the normative basis. Let TOPSTA_1 be the instance of top-stability saying that agents ℓ_3 and r_3 should be matched under profile $\boldsymbol{p}$ (because they rank each other in their top preferences). For this justification, the explanation is the set $\{\mathsf{LSS}_1, \mathsf{TOPSTA}_1\}$. The step-by-step argument is the following.

Again we start making a case distinction starting from LSS_1. For the relevant case, assume that the pair (ℓ_2, r_2) is not in the resulting matching. Now, by TOPSTA_1, we know that ℓ_3 and r_3 are matched together. Then, ℓ_2 cannot be assigned to r_3 either. Thus, the pair (ℓ_2, r_1) must be in the resulting matching. Therefore, ℓ_1 cannot be matched to r_1.

The explanation based on $\{\mathsf{LSS}, \mathsf{TOPSTA}\}$ is shorter and arguably more understandable than that one based on the set $\{\mathsf{LSS}\}$. Moreover the normative basis $\{\mathsf{LSS}\}$ is minimal, while $\{\mathsf{LSS}, \mathsf{TOPSTA}\}$ is not. ⊣

Observe that we can in principle, add more axioms to the normative basis, as it was done in Example 3.6 but this doesn't mean that instances of the added axioms will show up in the explanation and make it shorter. In fact, every non-trivial superset of a normative

[1] Because if both are in the resulting matching, then there would be two left agents (ℓ_1 and ℓ_2) that prefer each other's match.

basis of a justification is also a normative basis for some justification of the same feature.

Observation 3.5. If $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a justification for a problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$, then every non-trivial superset $\mathcal{A} \supseteq \mathcal{A}^N$ is a normative basis for a justification for the same problem. In fact, the pair $\langle \mathcal{A}, \mathcal{A}^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$.

Example 3.6 shows a situation where a justification with a minimal basis may not be better than one with a bigger normative basis. Hence, requiring minimality of the normative basis doesn't necessarily make the justification better.

3.4.2 Minimal Explanations

Now we discuss minimal explanations. The main purpose of a justification is to provide an explanation for humans (or the agents involved) to more easily understand why an outcome, or a specific feature of it, is a good compromise given the agents' preferences and a set of potentially interesting normative principles. Thus, it makes sense to look for explanations that are easier to understand than others. There is no unique way for an explanation to be more understandable. For example, the length of the explanation or the number of profiles involved can help in measuring how understandable the explanation is.

In this section we will propose a procedure to find a minimal explanation from a given one. We will show that it is always possible to obtain one, i.e., that the process is correct and always terminates.

In principle it is possible that for a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ there are different justifications based on the same set of normative principles. The choice of not requiring the explanation to be minimal for a justification to be well-defined makes it easier, for example, to join justifications. However, having minimal explanations can be useful to present the users with a step-by-step argument as clear as possible.

So, once we have a justification, we are interested in obtaining a *short* explanation. More precisely, a short explanation is an explanation that is minimal in size, i.e., that if we remove any of its elements, it can no longer explain the feature.

Given a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ and a justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ for it, a way to obtain a minimal explanation is to remove elements of the explanation one by one and check if the set still has explanatory power. We formalize this procedure in an algorithm, MIN-EXP.

MIN-EXP intuitively works as follows. It first creates two set variables: $\mathcal{A}$ in which it will store the minimal explanation; and $\mathcal{B}$, to keep track of the elements of the original explanation that it has already checked. Then, it runs a while loop that will stop being executed once $\mathcal{A} = \mathcal{A}^E$. That is, it will run as long as there are elements of the original

Algorithm 1 MIN-EXP

Input: A problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ and its justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$

1: $\mathcal{A} := \mathcal{A}^E$, $\mathcal{B} := \emptyset$
2: **while** $\mathcal{B} \neq \mathcal{A}^E$ **do**
3: Choose $a \in \mathcal{A}^E \setminus \mathcal{B}$
4: **if** $\mu(\boldsymbol{p}) \models \varphi$ for all $\mu \in \mathbb{I}(\mathcal{A} \setminus \{a\})$ **then**
5: $\mathcal{A} = \mathcal{A} \setminus \{a\}$
6: **end if**
7: $\mathcal{B} = \mathcal{B} \cup \{a\}$
8: **end while**
return $\mathcal{A}$

explanation set $\mathcal{A}^E$ that haven't been checked. In one execution of the loop, it chooses one element a from $\mathcal{A}^E \setminus \mathcal{B}$, i.e., an element from the original set that hasn't yet been checked. The set $\mathcal{A}^E \setminus \mathcal{B}$ is never empty if the while loop is entered. After choosing an element a, it verifies if the set $\mathcal{A} \setminus \{a\}$ is still an explanation for the feature. That is, if eliminating a from the current explanation set, stills results in an explanation. If it does, it updates the explanation set, by removing a from it. Finally, it updates the set of checked elements by adding a. When it has checked all the elements from the original explanation, i.e., when $\mathcal{B} = \mathcal{A}^E$, it returns the updated explanation set $\mathcal{A}$.

Observation 3.6. The algorithm MIN-EXP always terminates.

This is because inside the while loop, there is always one element from $\mathcal{A}^E$ added to $\mathcal{B}$ and the while loop stops being executed once these sets are equal. As there are a finite number of elements in $\mathcal{A}^E$, the loop runs only a finite number of times. Furthermore, every step inside the loop is finite. Hence, the algorithm terminates.

Proposition 3.7 (Correctness of MIN-EXP)**.** The algorithm MIN-EXP is correct, i.e., MIN-EXP always returns a minimal explanation.

Proof. First, notice that the starting set $\mathcal{A}^E$ is already an explanation.Then, the set $\mathcal{A}$, when it is first defined, is an explanation for the feature φ. Furthermore, every time the loop is executed, the set $\mathcal{A}$ is an explanation for φ, regardless of whether it was updated because of the check done inside the *if* clause.

Now, the explanation is minimal because once one element is removed, it still checks if all the remaining elements can be removed (and removes them if possible). So the set $\mathcal{A}$ is such that for every element $a \in \mathcal{A}$, the set $\mathcal{A} \setminus \{a\}$ is not an explanation, for every $a \in \mathcal{A}$. This implies that every proper subset of $\mathcal{A}$ is not an explanation anymore. Therefore, $\mathcal{A}$ is a minimal explanation. By Observation 3.6, MIN-EXP always terminates

and thus it returns some set $\mathcal{A}$. We conclude that MIN-EXP always returns a minimal explanation. □

A limit case is when the feature φ is a tautology.

Observation 3.7. If the feature φ is a tautology, then MIN-EXP returns the empty set.

That is because the set $\mathbb{I}(\emptyset)$ includes all matching mechanisms and every matching is a model for a tautology. So the algorithm will remove all the elements from the original explanation. This is still correct since a minimal explanation for a tautology is the empty set, as it was mentioned in the previous section (Section 3.3).

We have proven that the algorithm MIN-EXP always returns a minimal explanation. However, it doesn't always return the same explanation, or even an explanation of a certain size. Which explanation it returns depends on the order in which the elements of the original explanation set $\mathcal{A}^E$ are chosen. We illustrate this idea with Example 3.7.

Example 3.7. Suppose that the set $\mathcal{A}^E = \{a_1, a_2, a_3, a_4\}$ is the explanation set of a justification $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ for a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$. Furthermore, suppose that $\mathcal{A}^E$ is not minimal and that the minimal explanations for φ are $\{a_1, a_2, a_4\}$ and $\{a_3, a_4\}$.

Notice that if the algorithm chooses the elements of $\mathcal{A}^E$ in an order such that a_3 comes before a_1 and a_2, once it removes a_3 from the set it will not be able to remove neither a_1 nor a_2. In this case, MIN-EXP will return the set $\{a_1, a_2, a_4\}$. This corresponds to the path in blue in Figure 3.4.2. If on the contrary, we MIN-EXP chooses the elements of $\mathcal{A}^E$ in ascending order (a_1, a_2, a_3, a_4), it will remove elements as it is shown in the purple path in Figure 3.4.2. That is, it will remove a_1 first, then a_2 and it will return the set $\{a_3, a_4\}$. ⊣

3.5 Summary

In this chapter we defined a model for justifying partial outcomes of a matching mechanism under a certain profile and grounded in a set of normative principles. To do that we defined a propositional language to talk about partial matchings. Then, we showed that the language together with the model, behave as we would expect them to in correspondence with the interpretation of a justification. Furthermore, we showed how to obtain justifications from others, under certain conditions, like the compatibility of the feature or of the axioms in the normative bases. Finally, we discussed the property of minimality for both normative bases and explanations. We showed that a minimal normative basis

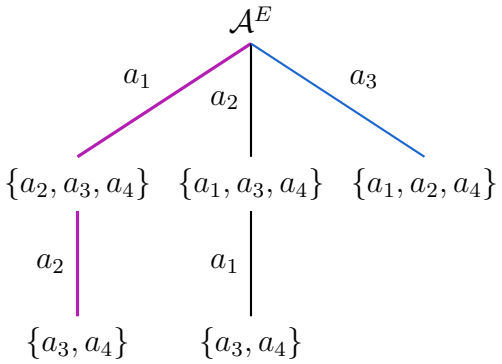

Figure 3.1: Different runs of MIN-EXP with input $\mathcal{A}^E$ as in Example 3.7. Each branch corresponds to a different run of the algorithm depending on the order in which the elements of $\mathcal{A}^E$ were chosen.

doesn't always result in a better justification in terms of the understandability of the explanation. For explanations, we described a process to obtain a minimal explanation and we showed that it is always possible to apply it.

4 | Automation via SAT Solving

In this chapter we will present one possible approach for the automation of the justification search. For this, we transform the problem into a propositional satisfiability problem and use SAT solvers to automatically solve it to then translate it back into our setting. This technique has been used in computational social choice for different purposes. First, we give an overview on how this technique has been used in the field (Section 4.1). Then we explain the general way in which it has been used (Section 4.2). After that, we explain what the approach will consist of in our specific problem (Section 4.3). We describe the outline of the program by explaining the encoding of the problem, the high level algorithm it follows and an implementation. Then, we show some examples of usage of the program and give an overview on its performance (Section 4.4). Finally, in Section 4.5, we discuss the approach, highlighting its advantages and proposing ideas to diminish its disadvantages.

4.1 SAT Solving in Computational Social Choice

Recently in the field of computational social choice, computer-aided methods have been used as tools for the further development of the theory of social choice. Some of the applications of computer-aided methods are, for example, the production of new proofs for known results or the search for new theorems. One of the tools used for these applications are SAT solvers. The technique of using SAT solvers to prove impossibilities in social choice was first used by Tang and Lin (2009). They used this approach to offer a new proof for Arrow's Theorem (Arrow, 1951). Their technique consists of a few steps that we will elaborate more on later. In their paper, Geist and Peters (2017) explain in depth this approach through an example of an impossibility result in voting theory.

Since the paper of Tang and Lin (2009), the use of SAT solvers has been applied in very diverse areas of computational social choice and game theory. Geist and Endriss (2011) used this method to verify and discover new theorems in the context of preference extensions. The approach was also used by Tang and Lin (2011) to discover theorems

in a special setting of game theory. Okada et al. (2019) used SAT solvers to discover new mechanisms for the location of a public good in a 2-by-2 grid and for proving an impossibility for the location of a public bad in a 2-by-3 grid. Brandl et al. (2015) prove an impossibility concerning the no-show paradox for set-valued voting rules using SAT solvers.

More related to this work, Endriss (2020), proves a *preservation theorem*, stating conditions for an axiomatic result, for one-to-one matching problems, to carry over to any setting with fewer agents. This means that one could prove an impossibility result by proving the impossibility for a fixed number of agents, and then the impossibility would automatically hold for any bigger number of agents (provided that the normative properties satisfy the conditions of the preservation theorem). Furthermore, Endriss proves two impossibility results for one-to-one matchings using this approach. He proves a base case with a small number of agents by means of a SAT solver and then uses the preservation theorem to validate the result for any greater instance.

4.2 The General Approach

In computer science, SAT is the decision problem where, given a propositional formula φ in CNF,[1] it answers whether there exists a truth assignment for the variables of φ that makes the formula true. SAT is known to be an NP-complete problem, which means that it belongs to the class and it is as hard as the problems that once presented with a witness for a "yes" answer, its correctness can be verified efficiently. In this particular case that means that once presented with a truth assignment for the formula, it can be verified in polynomial time that the assignment indeed makes the formula true. The problems in the class NP are also characterized as the decision problems that can be solved in polynomial time by a non-deterministic Turing machine. A good reference for this complexity class, and many others, can be found in the book by Arora and Barak (2009). In any case, problems in this class are know to be computationally *hard*. Nevertheless, nowadays there are algorithms that can solve most instances of SAT in a fairly efficient way.

SAT solvers are implementations of algorithms to solve the SAT problem. For some years now, the SAT Competition[2] has been taking place once a year. It is an event where implementors submit their solvers to compete by comparing the efficiency and correctness

[1] Conjunctive Normal Form (CNF) is a particular structure of a formula. It consists of conjunctions of clauses that are disjunctions of literals, where a literal is a variable or the negation of a variable. An example of such a formula is $\varphi = (x_1 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_4)$.

[2] http://www.satcompetition.org/

of their solvers. These solvers keep improving by means of new algorithms, heuristics and other kinds of optimizations. That is why the solving of some instances of SAT that seemed out of reach a few years ago, can be done in a regular fashion in the present day. For the interested reader in these topics, some of the algorithms and optimization techniques are nicely presented in the Handbook of Satisfiability (Biere et al., 2009).

Social choice and game-theoretic problems are suitable to be thought about from a computational point of view because the principles studied by the axiomatic method are easily expressible in simple mathematical notions, and thus they can be expressed in a formal language. Therefore, the SAT solving technique in these areas of study has been successful, especially in proving and discovering impossibilities. Recall that an impossibility result is a theorem stating that a mechanism for a collective decision cannot satisfy a certain set of axioms at the same time. In other words, the set of axioms is incompatible. For example, the Gibbard-Satterthwaite Theorem in voting (Gibbard, 1973; Satterthwaite, 1975) states that for an election with three or more voters, there is no voting rule that is surjective, strategyproof and not a dictatorship. In matching, Roth (1982) proved an impossibility result stating that there is no matching mechanism for a problem of size three or more that is both stable and strategyproof.

The approach, used by Tang and Lin (2009) and Geist and Endriss (2011), is based on an inductive argument that can be broken down as the following procedure. The first step is to encode a restricted version of the impossibility, with a *small* fixed number of agents and alternatives, and prove the result for this restricted setting with the help of the SAT solver. In other words, the first step consists of automatically proving a *base case*. In the next step the base case has to be translated back into a human-readable proof. And finally, it remains to prove an inductive lemma showing that the impossibility for the base case would carry over to any larger instance. This last step has to be done manually, and it usually represents the biggest challenge because it depends entirely on the particular result and it usually requires creativity. Only in the context of preference extensions (Geist and Endriss, 2011) and the context of matching (Endriss, 2020) meta-results have been proven, stating that if the axioms satisfy certain conditions of the way in which they can be expressed in a formal language, then any impossibility that can be proven for a fixed dimension of the problem can be extended to any bigger dimension.

We will now explain the high level idea of the first step of the process, the automatic part. Notice that this will apply to different social choice or game-theoretic settings where the axiomatic method is used. The idea is to encode as a propositional formula in CNF, α that the object in question[3] is well defined and that it satisfies the desired axioms. The

[3]The object can be, for example, a Social Choice Function in a voting setting; a coalition partition in a

encoding has to be done in such a way that the formula α is true if and only if there exists such an object with the set of properties that were encoded. This kind of encoding can be done because the axioms can be stated in a formal language. In his paper, Endriss (2020) defines a first-order language for axioms for matching mechanisms. So for instance, in a matching setting, once the dimension of the problem is fixed, the number of agents on both sides is finite and so is the number of profiles, so we can replace universal quantifiers by a *big* conjunction. This applies to other settings as well. Then, we can give α to a SAT solver and it will either output a model (a truth assignment for the variables of the formula) if α is satisfiable, or it will tell us that there doesn't exist any such model.

Though, notice that the sole output of the SAT solver is arguably not a valid formal mathematical proof of the statement even in the restricted case. Some arguments in favor of this idea are that the encoding may be flawed or that the probability that the SAT solver returns an incorrect answer is non-zero, maybe because of a mistake in the implementation or because of an error in the execution. Because of these reasons, it is good to still produce a human-readable proof of the base case. In the context of an impossibility theorem the extraction of a human-readable proof has still to be done in a semi-automated fashion, but once the proof is written in mathematical terms, it becomes more reliable. The semi-automated process of obtaining a human-readable proof has been done appealing to the concept of a *minimal unsatisfiable subset* (MUS). Given a set Δ of clauses, that are originally the conjuncts of the formula α, an MUS is an unsatisfiable subset of Δ such that all of its proper subsets are satisfiable. There exist computational tools that automatically extract an MUS from a set of clauses. We will talk about it more in depth in the following section. But then, for the proof of the base case of an impossibility, once we know that the formula representing the statement α is unsatisfiable, we can automatically extract an MUS and if it is small enough, it can be interpreted back and transformed into a human-readable proof.

We will make use of this ideas for the automation of the process of searching for a justification, but in a slightly different way.

4.3 Justification Search via SAT Solvers

In this section we will explain how to exploit the ideas presented and use SAT solvers to semi-automatically obtain a justification for a certain problem. The approach we use is not completely different to the one used to prove impossibilities. The encodings of both, the matching problem and the axioms are practically done in the same way. We also check

hedonic game; or a matching mechanism in the setting that concerns this work.

for sets of axioms to be non-trivial but we go further by encoding (negations of) feature formulas to search for normative basis and making use of MUSs to extract explanations. We explain this process in Section 4.3.2. But first, in Section 4.3.1 we describe how the encodings are done. Then, in Section 4.3.3 we explain how we implemented the procedures described in Section 4.3.2.

The high-level idea of this process goes as follows. Given a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$, we will establish a correspondence between a CNF formula α and the existence of a subset $\mathcal{A} \subseteq \mathbb{A}$ such that there is a justification for φ with normative basis $\mathcal{A}$. Furthermore, we will see how an MUS from a set of unsatisfiable formulas will correspond to an explanation set of a justification. We will also describe a procedure to semi-automatically obtain such justifications and we will prove its correctness.

We know that axioms for matching mechanisms can be expressed in a formal first-order language (Endriss, 2020). Furthermore, if the dimension of the matching problem is fixed, then we can express the axioms as propositional formulas and moreover, we can express them as CNF formulas.

For the automation of the justification search process one can use any SAT solver. We chose to use PICOSAT (Biere, 2008). In general, SAT solvers require the input formula to be in DIMACS format. In the DIMACS format, a formula in CNF corresponds to a list of lists of integers. The small lists represent the clauses and each integer represents a literal. Negative numbers are intended to represent negation. So for example, the list `[[1,2,3],[-2,-4],[1,-3]]` is in DIMACS format and corresponds to the propositional formula in CNF $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_1 \vee \neg x_3)$. We use a Python script to automatically generate the desired CNF formula in the correct format to be able to give it as input to the SAT solver. When presented with a formula in CNF, the SAT solver either returns `'UNSAT'` if the formula is unsatisfiable, or it returns a model, i.e., a truth assignment for the variables of the formula that make it true.

For the implementation of the justification search process, we were able to reuse some of the code written by Endriss (2019). We assume this code is saved under the name `matchsat.py`. The rest of the code that was written specifically for the search for justifications[4] and it is assumed to be saved under the name `just.py`.

[4] https://github.com/DanLK/MatchingJustificationSearch/tree/main/OneDrive%20-%20UvA/Thesis/Code/JustificationSearch

4.3.1 Encoding

Now we will see how we can encode in a formula the problem of whether a set of axioms can be the normative basis for a justification for a feature of a matching outcome. We will show how we constructed a Python script for automatic generation of the formulas. The process of writing the program mainly consists of three parts. The first part is the basic encoding of a matching problem in the Python script. The axioms we want to encode are properties about matching mechanisms, so we have to be able to talk about the elements that form a matching problem: the dimension n, the sets of agents L and R, agents' preference orders, possible profiles, among others. The second part is the automatic generation of formulas in CNF expressing that a matching mechanism satisfies a certain set of axioms. We will have the program generate one formula for every axiom that we are interested in and when we want to express that a mechanism satisfies a certain set of axioms we will take the conjunction of these formulas. The starting point for these two first steps is the code implemented by Endriss (2019) in his paper (Endriss, 2020) about the automatic proving of impossibility theorems in matching theory. There, he encodes all the basic elements of the matching model, some useful functions and some of the axioms that interest us. The code is used to prove impossibilities following the approach that we explained in Section 4.2. The last part of the encoding for our program consists of adding the information about the feature of a matching outcome (that we are interested in justifying), to the formula that we will use to feed the SAT solver. In the following we will describe the parts of the Python script that we used, the encoding of the axioms as CNF formulas in the DIMACS format passing through the intermediate step of expressing them in propositional logic, and the encoding of a feature. In the next sections (Section 4.3.2 and Section 4.3.3) we will explain how all this comes together as an implementation of a correct procedure for the search of a justification for a feature of a matching outcome in a given situation.

The model. We start by explaining the basic objects that constitute a matching problem that we need to represent in the program. Note that as the variables of a formula in the DIMACS format are integers, we need to encode every part of our problem as an integer. Notice that in a one-to-one matching problem with two-sided preferences (defined in Section 2.1), the dimension n of the problem, completely determines the sets L of left agents and R of right agents. This is because, abstractly, we only care about the size of these sets, so L consists exactly of agents $\ell_1, \ldots, \ell_n$ and a similarly, R consists of agents $r_1, \ldots, r_n$. Hence, in the encoding the agents will be identified by two integers: (i) their

type,[5] which will be 0 for left agents and 1 for right agents; and (ii) their index, which goes from 0 to $n-1$.

Then, each agent has to rank all the agents from the opposite group. As there are n of them, each agent can have $n!$ different preference orders. Then, all the different preferences that an agent can have are encoded as integers ranging from 0 to $n!-1$.

Once we have the preferences, we consider all different profiles. A preference profile consists of one preference order for each agent of each group. Thus, we have $(n!)^{2n}$ different preference profiles. Hence, we encode them as integer numbers that range from 0 to $(n!)^{2n}-1$.

In the program first the dimension n of the matching problem is fixed in a global variable `n` and then comes the implementation of functions that return all the possible indices for the agents (`allIndices()`), all the preferences (`allPreferences()`), and all the profiles (`allProfiles()`) as lists of integers, respectively. As an example, we have the function that returns all the preferences.

```
def allPreferences():
    return range(factorial(n))
```

Here `factorial` is a built-in Python function that can be imported from the `math` library, and it works as one would expect it to, in correspondence with the mathematical factorial function.

In the case of a matching problem of size 3, we have 6 different preference orders, so they are encoded with the numbers from 0 to 5; and the number of profiles is already quite big: $(3!)^6 = 46656$; so profiles are encoded with integers ranging from 0 to 46655.

The variables. For a matching problem of size n, we can completely describe a function that takes a preference profile and returns a subset of $L \times R$ by specifying, for every profile $\boldsymbol{p}$ and for every pair of agents $(\ell_i, r_j) \in L \times R$ whether they are assigned together in the outcome under $\boldsymbol{p}$. Thus, in the encoding, we will have one variable $x_{p,i,j}$ for every profile $\boldsymbol{p}$, every left agent $\ell_i \in L$ and every right agent $r_j \in R$ with the intended interpretation that variable $x_{p,i,j}$ will be true if and only if, under profile $\boldsymbol{p}$, the mechanism returns a matching that includes the pair (ℓ_i, r_j). Notice that when the matching problem dimension is n, the number of variables is equal to the number of profiles times the number of possible combinations of left and right agents. That is, $(n!)^{2n} \cdot |L \times R| = (n!)^{2n} \cdot n^2$. For instance, the number of variables when the matching problem is of size 3 is equal to $46656 \cdot 3^2 = 419904$.

In the Python program literals are encoded as integers. Recall that a literal is either a variable or a negation of a variable. So, it suffices to encode the variables as positive

[5] The type will only appear in some functions where it is actually needed.

integers and then represent the negation of the variable as the negative number that corresponds. For variable $x_{p,i,j}$ we assign the number $p \cdot (n \cdot n) + (i \cdot n) + j + 1$. This is an encoding choice that takes the variables in consecutive order corresponding to the profile, and agents in *lexicographic* order by fixing with strongest priority the profile number, then the left agent and then the right agent. It also adds one, shifting all the encodings by one to avoid assigning number zero to the first variable, since we need the additive inverse of all of them. Then, the negation of each variable is just encoded as the additive inverse of the integer that represents the variable. We call a literal positive if it is a variable and negative if it is the negation of a variable. Examples of some literals and their encodings when the dimension of the problem is 3 are given in Table 4.1.

Literal	...	$\neg x_{0,0,1}$	$\neg x_{0,0,0}$	$x_{0,0,0}$	$x_{0,0,1}$	$x_{0,0,2}$	$x_{0,1,0}$	$x_{0,1,1}$	...	$x_{1,0,0}$	...
Integer	...	-2	-1	1	2	3	4	5	...	10	...

Table 4.1: Encoding of some literals as integers for a matching problem of size 3.

The functions that do the encoding of positive and negative literals in the Python script look as follows.

```
def posLiteral(p, i, j):
    return p * (n * n) + (i * n) + j  + 1

def negLiteral(p, i, j):
    return (-1) * posLiteral(p, i, j)
```

The function `posLiteral(p,i,j)` returns the integer encoding literal $x_{p,i,j}$, while the function `negLiteral(p,i,j)` returns the encoding of literal $\neg x_{p,i,j}$.

The axioms. The program generates one formula expressing that a matching mechanism satisfies each of the axioms that we are interested in. Since the variables we are using completely describe a function that takes a profile and returns a subset of $L \times R$ by stating for every profile $\boldsymbol{p}$ and every pair of agents (ℓ_i, r_j), whether they are matched under profile $\boldsymbol{p}$, we still need to put a constraint on the kind of function that we are interested in. We only focus on matching mechanisms. That is, functions that return a proper matching. We call this property of such functions well-definedness. Thus, before encoding the axioms, we add a formula expressing the constraint that the function described by the variables is well-defined, or is a matching mechanism. In other words, well-definedness means that every left agent is matched with exactly one right agent and vice versa. This is equivalent to saying that every left agent is assigned to at least one right agent and every right agent is assigned to at most one left agent. This optimization of the encoding is done by Endriss (2019). The propositional formula φ_{MECH} expressing this property of a function that takes a profile and returns a matching is the following.

$$\varphi_{\mathsf{MECH}} = \bigwedge_{p\in\mathcal{P}_n}\left(\left(\bigwedge_{\ell_i\in L}\bigvee_{r_j\in R} x_{p,i,j}\right) \wedge \left(\bigwedge_{r_j\in R}\bigwedge_{\substack{\ell_{i_1}\in L,\\ \ell_{i_2}\in L}} (\neg x_{p,i_1,j} \vee \neg x_{p,i_2,j})\right)\right) \tag{4.1}$$

Notice that φ_{MECH} is in CNF. Now it is possible to use this formula to write a function in the Python script that generates the equivalent in DIMACS format. The part of the code that does this looks as in Listing 4.1.

```
def cnfMechanism():
    cnf = []
    for p in allProfiles():
        for i in allIndices():
            cnf.append([posLiteral(p,i,j) for j in allIndices()])
        for j in allIndices():
            for i1 in allIndices():
                for i2 in indices(lambda i2 : i1 < i2):
                    cnf.append([negLiteral(p,i1,j),
                        negLiteral(p,i2,j)])
    return cnf
```

Listing 4.1: Python script encoding that a matching mechanism is well defined.

In line 8 of Listing 4.1, the function `indices(condition)` is used. This function returns all the indices that satisfy the condition given as a parameter. In the case of how it's used in Listing 4.1, it returns the indices `i2` that are smaller than `i1`. In the Python script there are also implementations of similar functions that return preferences or profiles, given a certain condition. These are called `preferences(condition)` and `profiles(condition)`, respectively.

The process of encoding the axioms is similar. First, we express the axiom as a propositional formula in CNF and then it becomes easily translatable into a piece of code that will generate all the clauses automatically. Notice that the number of clauses of the formula φ_{MECH} grows very large with the dimension of the problem. For a matching problem of size 2, the formula consists of 64 clauses and when the size is 3, the formula already has 559872 clauses. This will be the case for almost all of the formulas that we care about.

We will now show some examples of axiom encodings. First, the axiom of stability (Definition 2.3), establishing that in the resulting matching there should be no blocking pair. We can rephrase the axiom in the following way. We will say that if there are agents $\ell_1 \in L$ and $r_1 \in R$ such that there exist other agents $\ell_2 \in L$ and $r_2 \in R$ and $\ell_1 \succ_{r_1} \ell_2$ and $r_1 \succ_{\ell_1} r_2$, then it cannot be the case that both ℓ_1 is matched with r_2 and r_1 is matched with ℓ_2. Because then ℓ_1 and r_1 would want to deviate from the matching to be together. Then, the following formula φ_{STA} is in CNF and it expresses that a matching mechanism

is stable.

$$\varphi_{\mathsf{STA}} = \bigwedge_{p\in\mathcal{P}_n} \bigwedge_{\ell_{i_1}\in L} \bigwedge_{r_{j_1}\in R} \bigwedge_{\substack{\ell_{i_2}\in L \text{ s.t.} \\ \ell_{i_1}\succ_{r_{j_1}}\ell_{i_2}}} \bigwedge_{\substack{r_{j_2}\in R \text{ s.t.} \\ r_{j_1}\succ_{\ell_{i_1}} r_{j_2}}} (\neg x_{p,1,2} \vee \neg x_{p,2,1}) \tag{4.2}$$

The Python code generating the formula in DIMACS format corresponding to φ_{STA} is shown in Listing 4.2. For a problem of size $n = 2$ the encoding of stability consists of 16 clauses, whereas when $n = 3$ it has 419904 clauses.

```
def cnfStable():
    cnf = []
    for p in allProfiles():
        for i1 in allIndices():
            for j1 in allIndices():
                for i2 in indices(lambda i2:prefers(1,j1,i1,i2,p)):
                    for j2 in indices(lambda j2:
                                      prefers(0,i1,j1,j2,p)):
                        cnf.append([negLiteral(p,i1,j2),
                                    negLiteral(p,i2,j1)])
    return cnf
```

Listing 4.2: Python script encoding stability.

Another interesting example is the encoding of strategyproofness. Recall that strategyproofness can be defined as a one-way principle (Definition 2.6) for either side of the market, or as a two-way principle (Definition 2.7) by taking both one-way principles together. We first illustrate the encoding of left-strategyproofness. The axiom (equivalently) states that for any left agent $\ell_i \in L$, and for every two preference profiles $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ that are ℓ_i-variants (Definition 2.5), for every right agent $r_{j_1} \in R$ and every other right agent $r_{j_2} \in R$ such that r_{j_2} is preferred over r_{j_1} by ℓ in the truthful profile $\boldsymbol{p}_1$, then either ℓ and r_{j_1} should not be matched together under $\boldsymbol{p}_1$ or ℓ and r_{j_2} should not be matched together under profile $\boldsymbol{p}_2$. The following formula φ_{LSP} encodes that a mechanism is left-strategyproof and it is in CNF.

$$\varphi_{\mathsf{LSP}} = \bigwedge_{\ell_i\in L} \bigwedge_{p\in\mathcal{P}_n} \bigwedge_{\substack{p' \text{ s.t. } p \text{ and } p' \\ \text{are } \ell_i-\text{variants}}} \bigwedge_{r_{j_1}\in R} \bigwedge_{\substack{r_{j_2}\in R \text{ s.t.} \\ r_{j_2}\succ^{p}_{\ell_i} r_{j_1}}} (\neg x_{p,i,j_1} \vee \neg x_{p',i,j_2}) \tag{4.3}$$

```
def cnfLeftStrategyProof():
    cnf = []
    for i in allIndices():
        for p1 in allProfiles():
            for p2 in iVariants(0, i, p1):
                for j1 in allIndices():
                    for j2 in indices(lambda j2 :
                                      prefers(0,i,j2,j1,p1)):
                        cnf.append([negLiteral(p1,i,j1),
                                    negLiteral(p2,i,j2)])
```

```
    return cnf
```

Listing 4.3: Python script encoding left-strategyproofness.

The code that generates the formula φ_{LSP} in DIMACS format is shown in Listing 4.3. When the matching problem is of size 2, the encoding of left-strategyproofnes is a formula with 32 clauses, and when the problem dimension is 3, the formula encoding it has 2099520 clauses. Then, for right-strategyproofness we can define a formula φ_{RSP} in an analogous way. We can also generate the code in a similar fashion and with those two formulas we define a formula φ_{SP} as the conjunction of φ_{LSP} and φ_{RSP} that encodes that a matching mechanism is two-way strategyproof and as the two conjuncts are in CNF, then φ_{SP} is also in CNF. In the program we can also just concatenate the two lists representing each one-way strategyproof principles to obtain the encoding of two-way strategyproofness. This is shown in Listing 4.4. The size of the formula encoding two-way strategyproofness when $n = 2$, considering that it is the conjunction of left and right-strategyproofness, is 64. Moreover, when the matching problem size is 3 the size of the formula encoding two-way strategyproofness is 4199040.

```
def cnfTwoWayStrategyProof():
    return cnfLeftStrategyProof() + cnfRightStrategyProof()
```

Listing 4.4: Python script encoding two-way strategyproofness.

As a last example of an encoding of an axiom, we take the axiom of left swap-stability (Definition 2.11), saying that for every profile, there is no pair of left agents such that they prefer each other's match in the outcome. A logically equivalent way of expressing this axiom is that for every profile $\boldsymbol{p}$, for every two left agents $\ell_1, \ell_2 \in L$ and every two right agents $r_1, r_2 \in R$, if ℓ_1 prefers (in profile $\boldsymbol{p}$) r_2 over r_1 and ℓ_2 prefers r_1 over r_2, then either ℓ_1 is not matched to r_1 or ℓ_2 is not matched to r_2. A propositional formula that expresses a statement equivalent to this axiom is φ_{LSS}, shown in Equation 4.4.

$$\varphi_{\mathsf{LSS}} = \bigwedge_{p \in \mathcal{P}_n} \bigwedge_{\ell_{i_1} \in L} \bigwedge_{\ell_{i_2} \in L} \bigwedge_{r_{j_1} \in R} \bigwedge_{\substack{r_{j_2} \in R \text{ s.t.} \\ r_{j_2} \succ^p_{\ell_{i_1}} r_{j_1} \\ \text{and } r_{j_1} \succ^p_{\ell_{i_2}} r_{j_2}}} (\neg x_{p,i_1,j_1} \vee \neg x_{p,i_2,j_2}) \tag{4.4}$$

The code that automatically generates formula φ_{LSS} in DIMACS format is shown in Listing 4.5.

```
def cnfLeftSwapStable():
    cnf = []
    for p in matchsat.allProfiles():
        for i1 in matchsat.allIndices():
            for i2 in matchsat.indices(lambda i2 : i2 < i1):
                for j1 in matchsat.allIndices():
                    for j2 in matchsat.indices(lambda j2 :
                                               matchsat.prefers(0,i1,j2,j1,p) and
                                               matchsat.prefers(0,i2,j1,j2,p)):
```

```
                    cnf.append([matchsat.negLiteral(p,i1,j1),
                                matchsat.negLiteral(p,i2,j2)])
    return cnf
```

Listing 4.5: Python script encoding left-swap-stability.

Recall that a formula in DIMACS format is a list of lists of integers, where the integers represent the literals and the small lists represent the clauses of the big conjunction. In Listing 4.6 an example of an axiom encoding in DIMACS format is shown. The formula corresponds to the axiom of left-swap-stability for a matching problem size of $n = 2$. It consists of 8 clauses. We only show the actual encoding for this axiom because it is quite small compared to the rest. However, when the problem size is 3, the encoding of left-swap-stability has 209952 clauses, so for reasons of space we don't show it.

```
>>> cnfLeftSwapStable()
[[-8, -5], [-11, -10], [-24, -21], [-27, -26],
[-40, -37], [-43, -42], [-56, -53], [-59, -58]]
```

Listing 4.6: Encoding of the axiom left-swap-stability in DIMACS format for a problem of size $n = 2$.

The first clause, `[-8,-5]`, is an instance of left-swap-stability making reference to a profile $\boldsymbol{p}$ (encoded as number 1), where agent ℓ_1 prefers agent r_2 over r_1, and on the contrary, agent ℓ_2 prefers agent r_1 over r_2. Both right agents prefer ℓ_2 over ℓ_1, but this is irrelevant since the axiom only considers the preferences of left agents. Taking into account the encoding of the agents (with indices from 0 to $n-1$), the literal -8 corresponds to the variable $\neg x_{1,1,1}$ and literal -5 corresponds to $\neg x_{1,0,0}$. Hence, the clause `[-8,-5]` corresponds to the instance of left-swap-stability saying that in profile $\boldsymbol{p}$, it should not be the case that agents ℓ_1 and r_1 are matched and ℓ_2 and r_2 are matched at the same time. The rest of the clauses can be interpreted in a similar way.

In Table 4.2 a summary of the functions generating axiom encodings that were implemented is provided. The first column corresponds to the name of the axiom, the second to the name of the function that generates the encoding, and the third to the name of the file in which the function was implemented.

Notice that when encoding the axioms as propositional formulas in CNF, we are expressing them in a formal language. Here the notion of instance discussed in Section 3.2 becomes more clear. We know that axioms for matching mechanisms can be expressed as formulas in a first-order language (Endriss, 2020) and that the general procedure of transforming them into propositional formulas consists of constructing a big conjunction which conjuncts are *instantiations* of the variables that were in the scope of the quan-

[6]In the file `matchsat.py` the function is called `cnfGenderIndifferent()`, but to make it consistent with our naming of the axiom, we use the same code but under the name `cnfGroupIndifferent()` in the file `just.py`.

AXIOM NAME	FUNCTION NAME IN PYTHON SCRIPT	FILE
Stability (STA)	`cnfStable()`	`matchsat.py`
Top-stability (TOPSTA)	`cnfTopStable()`	`matchsat.py`
Left-strategyproofness (LSP)	`cnfLeftStrategyProof()`	`matchsat.py`
Right-strategyproofness (RSP)	`cnfRightStrategyProof()`	`matchsat.py`
Strategyproofness (SP)	`cnfTwoWayStrategyProof()`	`matchsat.py`
Group-indifference (GI)	`cnfGroupIndifferent()`	`matchsat.py`[6]
Peer-indifference (GI)	`cnfPeerIndifferent()`	`matchsat.py`
Left top-rewarding (LTR)	`cnfLeftFavorite()`	`just.py`
Right top-rewarding (RTR)	`cnfRightFavorite()`	`just.py`
Top-rewarding (TR)	`cnfTopRewarding()`	`just.py`
Left-swap-stability (LSS)	`cnfLeftSwapStable()`	`just.py`
Right-swap-stability (RSS)	`cnfRightSwapStable()`	`just.py`
Swap-stability (SS)	`cnfSwapStable()`	`just.py`
No bottoms (NBOT)	`cnfNoBottoms()`	`just.py`

Table 4.2: Names of the functions implemented to generate axiom encodings.

tifiers in the first-order formula. We know that this is feasible, i.e., that in results in a well-formed formula, because once the dimension of the matching problem is fixed, the objects over which we quantify become finite. These objects are, for example, agents or profiles. For example, we can express a principle saying that some property that depends on the profile holds for every profile, as the conjunction of one instance of the property for each profile. In this sense, we can say that the encoding of an axiom in a propositional formula can be seen as a collection of its instances, each instance being one conjunct of the formula. After this first step, the propositional formula is already a conjunction of formulas but at this point we cannot really say anything about the shape of the conjuncts; in principle they could also be conjunctions. However, we know that any propositional formula can be transformed into a formula in CNF. But in that process we could lose the property that every conjunct (clause) of the formula corresponds exactly to one instance. This correspondence between instances of an axiom and the clauses of its encoding as a CNF formula is something desirable for the justification search, as we will see in the following section.

We have seen how an axiom can be encoded as a formula in CNF. We also mentioned that it is possible to encode that a mechanism satisfies a certain set of axioms as a CNF formula by taking the conjunction of the encodings of each axiom of the set. As we will be interested in whether these kinds of formulas are satisfiable, it is important to note that a formula in CNF is satisfiable if and only if all of its clauses are satisfied by the same truth assignment of its variables. This is why we will often only be interested in the set

of clauses of a formula. A set of propositional formulas is said to be satisfiable if all of its formulas are satisfiable by the same truth assignment. Thus, we can identify a CNF formula encoding that a mechanism satisfies a set of axioms with its set of clauses. In this regard, for practicality and when the context is clear, for a set of axioms $\mathcal{A}$, we will often refer to the set of clauses of the CNF formula encoding the axioms in $\mathcal{A}$ as its *encoding set* and we will denote it by Δ.

As we have seen, it's not necessarily the case that under the way we encoded the axioms, we always obtain formulas which clauses correspond exactly to instances of the axiom they are an encoding of. For example, consider the encoding for the principle of being a well-defined matching mechanism φ_{MECH} (Formula 4.1). In this case, when the matching problem is of size $n = 2$, an example of an encoding of an instance of this principle would be

$$(x_{p,1,1} \vee x_{p,1,2}) \wedge (\neg x_{p,1,1} \vee \neg x_{p,2,1}),$$

when we fix some profile $\boldsymbol{p}$, left agent ℓ_1, right agent r_1, and pair of left agents ℓ_1 and ℓ_2. But this instance is clearly not expressed by a single clause. In contrast, take the encoding φ_{STA} of the axiom of stability. When the dimension of the matching problem is $n = 2$, one instance of this axiom is encoded as the formula

$$(\neg x_{p,1,2} \vee \neg x_{p,2,1}),$$

where the profile $\boldsymbol{p}$ consists of both left agents preferring agent r_1 over agent r_2 and both right agents preferring agent ℓ_1 over ℓ_2, and the blocking pair being (ℓ_1, r_1). So, in the case of this axiom, there is a direct correspondence between its instances and the clauses of its encoding.

Although it is not always the case that this correspondence exists, we can say that it exists in most cases. For the axioms that we implemented, it is always the case except for peer-indifference. As we saw, it also doesn't hold for the principle of well-definedness. The correspondence between clauses and instances will become important when we want to extract an explanation for a justification. Ways to circumvent this issue will be discussed in the following section.

The features. Features of a matching outcome under a certain profile can also be encoded as propositional formulas using the same variables that we have defined and thus, they can be transformed into formulas in DIMACS format. For instance, a basic feature such as the pair (ℓ_i, r_j) being part of the outcome of the mechanism under certain profile $\boldsymbol{p}$ corresponds exactly to the formula $x_{p,i,j}$. Then, as all the features we care about are boolean combinations of these basic features, and every propositional formula can be

expressed as a CNF formula, we can be sure that any feature can be expressed as a CNF formula with these variables and hence, written in DIMACS format.

So far we have seen how to encode axioms and features as propositional formulas and transform those into formulas in the correct format for the input of a SAT solver, namely formulas in DIMACS format.

4.3.2 The Algorithm

In this section we will layout a procedure to search for a justification for a feature in a given situation. The correctness of this procedure relies on the fact that we can encode axioms of matching mechanisms as CNF formulas and that we can do the same with negations of feature formulas. Throughout this section we assume that the encoding of an axiom is such that a clause corresponds to an instance (recall the discussion in Section 4.3.1). The high level idea for the procedure is that, given a justification problem, we will check for every subset of the corpus of axioms, whether it can constitute a normative basis for a justification for the feature. Once a normative basis is found, we will extract an explanation taking advantage of the concept of an MUS. This argument depends on the specific way that we encoded the axioms and the features, so from now on when we refer to an encoding, we assume that it is done in the way we explained in Section 4.3.1. Before formally describing this procedure as an algorithm, we prove that there is a correspondence between the satisfiability of a set of clauses and the existence of a normative basis for a justification. We also prove how the concept of an MUS connects with an explanation for a feature of a matching outcome. For this, we have to make a rather strong assumption but we will discuss later why the procedure still works in most of the cases.

First we establish the correspondence between an unsatisfiable set of clauses and the existence of a normative basis for a justification. Suppose that we have a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ and that there is a subset of axioms $\mathcal{A} \subseteq \mathbb{A}$ such that $\Delta \cup \Phi$ is unsatisfiable while Δ is satisfiable, where Δ is the encoding set of $\mathcal{A}$ and Φ is the set of clauses encoding the negation $\neg\varphi$ of the feature. Then, we can say that the set $\mathcal{A}$ is a normative basis of a justification for φ. Intuitively, this is because Δ being satisfiable means that there is at least one mechanism satisfying the axioms in $\mathcal{A}$, so the set of axioms is non-trivial. And $\Delta \cup \Phi$ being unsatisfiable means that every mechanism that satisfies the axioms in $\mathcal{A}$ cannot return an outcome that doesn't have feature φ. We formally prove this in Proposition 4.1.

Proposition 4.1. Let $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ be a justification problem. Let $\mathcal{A} \subseteq \mathbb{A}$ be a subset of the corpus of axioms. Suppose that Δ is the set of clauses of the CNF formula encoding

the axioms in $\mathcal{A}$. Furthermore, suppose that Φ is the encoding as a CNF formula of the negation $\neg\varphi$ of the feature. If the set $\Delta \cup \Phi$ is unsatisfiable while Δ is satisfiable, then $\mathcal{A}$ is the normative basis for a justification of the problem.

Proof. It suffices to prove that there exists an explanation $\mathcal{A}^E$ such that $\langle \mathcal{A}, \mathcal{A}^E \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$. To prove this, take $\mathcal{A}^E$ as the set of instances of all axioms in $\mathcal{A}$. We have to prove that $\langle \mathcal{A}, \mathcal{A}^E \rangle$ is a justification for the problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$.

First we are going to prove that the explanatoriness condition is satisfied. Let $\mu \in \mathbb{I}(\mathcal{A}^E)$. Recall that we assumed that there is a correspondence between instances of axioms and clauses of their representation as formulas in CNF. So, the set of clauses encoding the axiom instances in $\mathcal{A}^E$ corresponds exactly with the encoding set Δ of $\mathcal{A}$.

We know that Δ is satisfiable, so every truth assignment for the variables that makes the clauses in Δ true, describes a mechanism in $\mathbb{I}(\mathcal{A}^E)$. So there is a truth assignment for the variables describing μ.

As $\Delta \cup \Phi$ is unsatisfiable, there is no truth assignment for the variables that makes true all the clauses in Δ and the clauses in Φ at the same time. In particular, the truth assignment that describes μ, cannot make all the clauses in Φ true. Then it must make true the negation of the CNF encoding the negation $\neg\varphi$. So, the truth assignment should make true the encoding of the feature φ. This means that the mechanism μ is such that under profile $\boldsymbol{p}$, it returns a matching that has feature φ. That is, $\mu(\boldsymbol{p}) \in \text{Mod}(\varphi)$. Hence, the set $\mathcal{A}$ can explain feature φ and explanatoriness holds.

The relevance condition holds because of the choice of $\mathcal{A}^E$. Adequacy holds by assumption: $\mathcal{A} \subseteq \mathbb{A}$. And non-triviality follows from the fact that Δ is satisfiable.

Hence, there exists a justification grounded in the set $\mathcal{A}$ for the problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$. □

This means that with the encoding of the axioms and features that we described in Section 4.3.1, we can automate the search, from a corpus of axioms, for a subset of them that is a basis for a justification for the feature in question, if there exists one. We formalize this procedure in Algorithm 2. The input for the algorithm is a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$. Recall that for a set of axioms $\mathcal{A}$, we use Δ to denote the set of clauses of the encoding of the axioms in $\mathcal{A}$; and we use Φ to denote the set of clauses encoding of the negation $\neg\varphi$ of the feature.

Clearly, BASIS-SEARCH terminates, since the number of subsets of a set is finite. Moreover, if there is a subset of the corpus that constitutes a normative basis for a justification for the feature, then BASIS-SEARCH finds it. This follows from Proposition 4.1. Therefore, the algorithm BASIS-SEARCH is correct.

Algorithm 2 BASIS-SEARCH

Input: A justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$
Let Φ be the encoding of $\neg\varphi$.

1: **for** $\mathcal{A} \subseteq \mathbb{A}$ **do**
2: $\quad\Delta :=$ the encoding set of $\mathcal{A}$
3: $\quad$**if** $\Delta \cup \Phi$ is unsatisfiable while Δ is satisfiable **then**
4: $\quad\quad$**return** $\mathcal{A}$
5: $\quad$**end if**
6: **end for**
7: **return** "There is no justification based on axioms from $\mathbb{A}$."

This means that provided that we can automate the check of the condition in line 2 of Algorithm 2, we can also automate the search for a normative basis. But with the encoding discussed in Section 4.3.1, we can check this automatically by just giving the appropriate formula to a SAT solver. Notice that this is nevertheless not feasible for every problem size, but we leave this discussion for later.

We have established a correspondence between a formula in CNF encoding a set of axioms and the negation of the feature we are trying to justify, and the existence of such justification. Furthermore, we know that it is possible to do this in an automatic fashion. But in order to obtain a proper justification, we still need to extract an explanation, which in practice is arguably the most important part of it. In the following we explain how this is possible using the concept of an MUS of a set of unsatisfiable clauses. For this procedure, we need to restrict the class of features to those for which the encoding is a single clause. We make the following assumption.

Assumption 4.1. The negation of a feature can be encoded as a single clause. That is, we only consider justification problems $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ such that $\neg\varphi$ can be encoded as a propositional formula with exactly one clause.

Suppose that we have a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$ and that we have found a subset of axioms $\mathcal{A} \subseteq \mathbb{A}$ that is a normative basis for a justification for φ. Again, we use Δ to denote the set of clauses of the CNF formula encoding the axioms in $\mathcal{A}$, and this time we use α for the encoding of the negation $\neg\varphi$ of the feature, since it is only one clause (Assumption 4.1). Observe that the proof of Proposition 4.1 implies that the set of all instances of $\mathcal{A}$ can already constitute an explanation, but as we discussed in Section 3.4.2, we are interested in *small* explanations in practice, so it wouldn't make sense to just take the whole set of instances of $\mathcal{A}$. In any case, the key idea for $\mathcal{A}$ to be relevant as a justification for φ is that the set Δ is satisfiable and $\Delta \cup \{\alpha\}$ is unsatisfiable . Recall that we supposed that the axioms in $\mathcal{A}$ are such that each clause of their encoding corresponds

to one instance. Thus, if we find a subset Σ of $\Delta \cup \{\alpha\}$ that is still unsatisfiable, then $\Sigma \setminus \{\alpha\}$ will correspond to an explanation for the justification. That is because first, as Δ is satisfiable, any subset of it is still satisfiable, so for Σ to be satisfiable it must contain $\{\alpha\}$ and this, following the same reasoning as in the proof of the explanatoriness condition in Proposition 4.1, implies that it can explain the feature φ. And second, it will correspond to a set of instances of the axioms in $\mathcal{A}$ because it is just a subset of the clauses and we supposed that each clause is an instance of an axiom. This is where the concept of an MUS comes in handy. Recall that an MUS is a set of clauses that is unsatisfiable and every proper subset of it is still satisfiable. So if we take Σ to be an MUS of $\Delta \cup \{\alpha\}$, it will have the properties that we are looking for and it will usually be smaller than the whole set $\Delta \cup \{\alpha\}$. Furthermore, there are tools to automatically compute an MUS from a set of unsatisfiable clauses, so this helps achieve the goal of automating the process. We will show how to do this in a semi-automatic way but first we prove that this argument is indeed correct (Proposition 4.2).

Proposition 4.2. Given a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$, suppose that there is a set of axioms $\mathcal{A} \subseteq \mathbb{A}$ that is a normative basis for a justification for φ. Let Δ be the encoding set of $\mathcal{A}$ and let α be the encoding of the negation $\neg\varphi$ of the feature. Assume that every axiom $A \in \mathcal{A}$ is such that a clause of its encoding as a CNF formula corresponds exactly to an instance of A. Finally, consider Assumption 4.1. Then, if Σ is an MUS of the set $\Delta \cup \{\alpha\}$, the following conditions hold:

(i) $\alpha \in \Sigma$; and

(ii) $\Sigma \setminus \{\alpha\}$ corresponds to an explanation for the matching outcome having feature φ.

Proof. First we prove (i). We know that $\Sigma \subseteq (\Delta \cup \{\alpha\})$. So if $\alpha \notin \Sigma$, then $\Sigma \subseteq \Delta$. But Δ is satisfiable, so every subset of Δ is satisfiable too. Hence, Σ would be satisfiable, which contradicts the fact that Σ is an MUS. Therefore $\{\alpha\} \subseteq \Sigma$. Notice that here Assumption 4.1 is essential because if the encoding of the negation of the feature consisted of more than one clause, we could not guarantee all of them being part of the MUS.

Now we prove (ii). First notice that each clause in the set $\Sigma \setminus \{\alpha\}$ corresponds to an instance of an axiom of $\mathcal{A}$. This holds by assumption. Let's denote the set $\Sigma \setminus \{\alpha\}$ by Σ'. Now let $\mathcal{A}^{\Sigma'}$ denote the set of instances of axioms represented by the clauses in Σ'. We have to prove that $\langle \mathcal{A}, \mathcal{A}^{\Sigma'} \rangle$ is a justification for $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$.

We start by showing that the explanatoriness condition holds. Let $\mu \in \mathbb{I}(\mathcal{A}^{\Sigma'})$. The set Σ' is satisfiable because $\Sigma' \subseteq \Delta$. So, every truth assignment for the variables of the clauses in Σ' that makes all the clauses true at the same time, is such that it describes one

mechanism in $\mathbb{I}(\mathcal{A}^{\Sigma'})$. So, let ν be a truth assignment for the variables of the formulas in Σ' such that it defines the mechanism μ (and makes true all the clauses in Σ'). Then ν cannot make true α because $\Sigma = \Sigma' \cup \{\alpha\}$ is unsatisfiable. So, ν has to make true $\neg\alpha$, which corresponds to the encoding of the feature φ. Thus, μ is such that it always returns a matching with feature φ. Hence, $\mu \in \text{Mod}(\varphi)$. Therefore, explanatoriness is satisfied.

Relevance is satisfied because we assumed that all the clauses in Σ' correspond to instances of the axioms in $\mathcal{A}$. So, $\mathcal{A}^{\Sigma'} \triangleleft \mathcal{A}$. Adequacy holds by assumption: $\mathcal{A} \subseteq \mathbb{A}$. And finally, non-triviality follows from the assumption that Δ is satisfiable. Hence, $\langle \mathcal{A}, \mathcal{A}^{\Sigma'} \rangle$ is a justification for the feature φ. □

There are three main things to note about Proposition 4.2 that will be relevant when we use it in practice. First, when we extract an MUS from a set of unsatisfiable formulas, what we get is another set of formulas. In practice, an *interpretation* of the latter will still need to be done. Second, in Proposition 4.2, we assumed that every axiom from the normative basis was such that its instances had a direct correspondence with the clauses of its encoding. This is arguably a strong assumption since we saw that this is not always the case in the way that we encoded the axioms. Although we will see that in practice, this is not so relevant since then we will obtain an explanation consisting of (possibly) weaker logical statements. This could even be better than obtaining an explanation consisting exactly of instances of axioms because the set of weaker logical statements can result in an easier explanation for a human to understand. And moreover, we can always do the translation from the MUS back to human-readable statements in such a way that if the clauses appearing in the MUS are not complete instances of the axioms, we add the missing part of the instance.

The third thing to note in Proposition 4.2 is regarding Assumption 4.1: the restriction to the class of features which negation's encoding consists exactly of one clause. The encoding of the negation $\neg\varphi$ of the feature consisting of more than one clause, say $\alpha_1, \ldots, \alpha_m$, is problematic because if that were the case, we could not guarantee that the extraction of an MUS will result in a proper explanation. In that case the source of unsatisfiability of $\Delta \cup \{\alpha_1, \ldots, \alpha_m\}$ would not necessarily be the whole set $\{\alpha_1, \ldots, \alpha_m\}$. Then, when extracting an MUS Σ, it would not be guaranteed that all of these clauses are part of it. This would break condition (i) of the Proposition. An MUS extracted in that case would then correspond to an explanation of something that would not exactly be the feature that we wanted to justify in the first place.

However, the restriction to this class of features is arguably not too bad after all. Note that the features of which negations are encoded as CNFs with exactly one clause

are conjunctions of literals of the feature language (defined in Section 3.1). Recall that the variables of the feature language are of the form $(i \rightleftarrows j)$, where $1 \leq i, j \leq n$ and that a literal is either a variable or a negation of a variable. Hence, these are features expressing that pairs of agents are or aren't part of the outcome. This class of features is still interesting and it covers many relevant cases. Nevertheless, there are still some interesting cases that are in principle expressed, for instance, by a disjunction of literals. For example, in a matching problem of dimension $n = 4$, if we wanted to search for a justification for the feature $\varphi = (1 \rightleftarrows 1) \vee (1 \rightleftarrows 2)$, i.e., that left agent ℓ_1 is either assigned right agent r_1 or r_2. In principle by Assumption 4.1, it would not be guaranteed that our procedure for finding an explanation for φ works, even when we have found a normative basis, because the encoding of the negation $\neg\varphi$ of the feature is $\neg x_{\boldsymbol{p},1,1} \wedge \neg x_{\boldsymbol{p},1,2}$, where $\boldsymbol{p}$ is the profile of the justification problem we are working with. So, it consists of more than one clause. But in cases like this we can take advantage of the feature language and take an equivalent formula, say $\psi = \neg\,(1 \rightleftarrows 3) \wedge \neg\,(1 \rightleftarrows 4)$. The encoding of the negation $\neg\psi$ has the desired form: it consists of exactly one clause, namely $x_{\boldsymbol{p},1,3} \vee x_{\boldsymbol{p},1,4}$, and by Corollary 3.1, we can search for a justification for ψ because if we find one it would also be a justification for φ. Thus, even though the result only holds for a restricted class of features, in practice we can apply the procedure to (almost) all kinds of features.

Returning to the description of a procedure for the automatic search for a justification, we have shown that once we have automatically found a normative basis, we can extract an MUS that corresponds to an explanation of the justification and that is hopefully smaller than the whole set of axioms forming the normative basis.

Now we bring Proposition 4.1 and Proposition 4.2 together to describe an algorithm that, given a justification problem, will return a normative basis together with a set of clauses corresponding to an explanation, that will form a justification for the input justification problem. This procedure is formalized in the algorithm JUST-SEARCH (Algorithm 3). Notice that since we are making use of the fact proven in Proposition 4.2, we are restricted to the class of features specified in Assumption 4.1. Algorithm JUST-SEARCH takes as input a justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$. First the algorithm calls BASIS-SEARCH with the same problem as input. That is, it first searches among the axioms in the corpus for a normative basis for a justification for the problem. If it finds a basis $\mathcal{A}$, then it extracts an MUS from its encoding set together with the encoding of the negation of the feature, i.e., $\Delta \cup \{\alpha\}$. Finally, it returns the MUS together with the normative basis. If BASIS-SEARCH returns that there is no justification, JUST-SEARCH returns this information too.

Not that the algorithm JUST-SEARCH always returns what corresponds to a justifi-

Algorithm 3 JUST-SEARCH

Input: A justification problem $\langle \mathbb{A}, \boldsymbol{p}, \varphi \rangle$
Let α be the encoding of $\neg\varphi$

1: **if** BASIS-SEARCH returns a normative basis $\mathcal{A}$ under the same input **then**
2: $\quad \Delta :=$ the encoding set of $\mathcal{A}$
3: $\quad$ Extract MUS Σ from $\Delta \cup \{\alpha\}$
4: $\quad$ **return** $\mathcal{A}$, Σ
5: **end if**
6: **return** "There is no justification based on axioms from $\mathbb{A}$."

cation for a problem, if there exists one. It returns the normative basis $\mathcal{A}^N$ and a set of clauses that are the encoding of an explanation $\mathcal{A}^E$ such that the pair $\langle \mathcal{A}^N, \mathcal{A}^E \rangle$ is a justification for φ. The correctness of the algorithm follows directly from Proposition 4.1 and Proposition 4.2. We have proven that there is a correct procedure to find a justification for a feature of a matching outcome, whenever one exists.

4.3.3 Implementation

We will now illustrate how the justification search can be performed semi-automatically using SAT solvers. For this, we extend the Python script described in Section 4.3.1. Although both, the procedure described as algorithm BASIS-SEARCH and the extraction of the MUS can be fully automated, one last step for obtaining a human-readable explanation remains to be done, namely the inspection and translation of the MUS, which is mostly done manually.

The extension of the Python script only covers the process of searching for a normative basis for a justification. The rest is done with assistance of the computer but requires more involvement of the user. So first we explain the implementation of the algorithm BASIS-SEARCH.

Recall that in the first part of the Python script we automated the generation of propositional formulas in CNF corresponding to axioms of matching mechanisms and features of matching outcomes. We will use these encodings to generate the input of the implementation of the BASIS-SEARCH algorithm.

We will describe the implementations of two functions, called `allBasesSearch` and `basisSearch`, respectively. The function `allBasesSearch` takes as input the encodings of both a set of axioms and the negation of the feature and it returns all the possible subsets of the corpus that are normative bases for the justification we are searching for; whereas the function `basisSearch` is the implementation of the algorithm BASIS-SEARCH, i.e., returns the first normative basis that it finds for the justification. Actually,

`basisSearch`, takes the same input as `allBasesSearch` plus a string that will be the name of the file in which the formula produced will be saved. We present both functions here because in practice, we are mostly interested in obtaining one normative basis with the function `basisSearch`, but its code is actually an adaptation coming from the more general version, `allBasesSearch`.

Before explaining how the functions `allBasesSearch` and `basisSearch` work, we need an auxiliary function `isJustification`, that checks, given the encoding of a set of axioms and the encoding of the negation of a feature, whether exactly those axioms are a normative basis for a justification for the feature. The definition of the function `isJustification` is shown in Listing 4.7. The function takes a *set* Δ of encoded axioms and the encoding of the negation of the feature and returns `True` if and only if the axioms corresponding to Δ are a normative basis for a justification for the feature.

The input for the function `isJustification` consists of `axioms`, a list of axioms in DIMACS format and `nfeature`, the encoding of the negation of the feature, also in DIMACS format.

```
def isJustification(axioms,nfeature):
    cnf = list(chain(*axioms)) + matchsat.cnfMechanism()
    if solve(cnf) == 'UNSAT':
        return False
    else:
        cnf += nfeature
        return solve(cnf) == 'UNSAT'
```

Listing 4.7: Python script for the function isJustification

The function `isJustification` first flattens the list of axioms, so that it gets transformed into a list (`cnf`) with all their clauses and it adds to this formula the constraint that the mechanism is well-defined. Then it checks whether the set of axioms is satisfiable. If it's not, then it cannot be a normative basis. If the set of axioms together with the constraint that the mechanism is well-defined is satisfiable, it includes in the set of clauses the clause for the negation of the feature and finally, it checks whether the formula `cnf` is satisfiable. Both satisfiability checks are done inside the function using the SAT solver LINGELING[7] that comes with a Python module, PYLGL, that allows to call the SAT solver directly from Python via the function `solve`. The function `solve` works as one would expect, it takes a list of lists, corresponding to a formula in CNF in DIMACS format and it either returns `'UNSAT'`, if the formula is unsatisfiable, or it returns a model, i.e., a truth assignment for the variables of the formula, that makes the formula true. So, the function `isJustification` returns the value `True` if and only if the axioms are a

[7] http://fmv.jku.at/lingeling/

normative basis for a justification of the feature.

In the function `isJustification`, the part of the input corresponding to the axioms (`axioms`) consisted only of the axioms encoded, i.e., a list of lists of clauses. In the case of `allBasesSearch` and `basisSearch`, we provide this part of the input with some extra information. Notice that with the encoding of the axioms in formulas in DIMACS format, it would be very hard if the function returned a set of clauses, to figure out exactly to which subset of axioms they correspond. So what we do is include the *names* of the axioms in the input. So, `axioms` is now a list of two-element lists. Each of them has as first element the *name* of the axiom and as second element the formula in DIMACS encoding it. The other part of the input, `nfeature` is still the one-clause encoding of the negation of the feature, so a list of one element which is also a list. The code for the function `allBasesSearch` is shown in Listing 4.8.

```
def allBasesSearch(axioms, nfeature):
    search_space = subsets(axioms)
    search_space.remove([])
    bases = []
    for set_of_axioms in search_space:
        result = []
        axioms = []
        for axiom in set_of_axioms:
            axioms += [axiom[1]]
        if isJustification(axioms,nfeature):
            for axiom in set_of_axioms:
                result.append(axiom[0])
            bases.append(result)
    return bases
```

Listing 4.8: Python code to obtain all possible normative bases

The function `allBasesSearch` searches over all subsets of the axioms and for each set, it checks whether it is a normative basis using the function `isJustification`. If it is, it appends the subset to the list `bases`, and when it finishes searching over all the subsets, it returns the list `bases`, where all the normative bases were stored. Thus, `allBasesSearch` indeed computes a list with all the subsets of the corpus that are a normative basis.

Note that we use a function `subsets` to generate the search space. As it is expected, it computes all the possible subsets[8] of the elements of a list. The search space grows exponentially with the size of the corpus. Recall that for a set with k elements, the number of subsets is 2^k. So, if the corpus is too big, searching over all possible subsets of

[8]The data structures that the function generates are lists but we call them subsets because it is conceptually what they represent.

axioms may become infeasible. Hence, in practice we are only interested in obtaining one normative basis.

Now, the function `basisSearch` corresponds to the implementation of the algorithm BASIS-SEARCH (Algorithm 2) and the code is shown in Listing 4.9. This function does the task of searching for a normative basis in the same way as `allBasesSearch` does but it differs in that `basisSearch` stops once it has found one basis and returns it. The function `basisSearch` only has to search over all the subsets in case there is no justification (so, it doesn't stop earlier). Another difference is in the parameters it receives. Similarly to `allBasesSearch`, the function `basisSearch` receives a set of encoded axioms with names and the encoding of the negation of a feature; but `basisSearch` also receives a string `filename`, that will be the name for the file in which the CNF formula corresponding to the encoding of the axioms is going to be saved. This last part will be discussed later, but the formula is saved in a text file so we can extract the MUS from there.

```
def basisSearch(axioms, nfeature, filename):
    search_space = subsets(axioms)
    search_space.remove([])
    for set_of_axioms in search_space:
        result = []
        cnf = matchsat.cnfMechanism()
        axioms = []
        for axiom in set_of_axioms:
            axioms += [axiom[1]]
        if isJustification(axioms,nfeature):
            for axiom in set_of_axioms:
                cnf += axiom[1]
                result.append(axiom[0])
            cnf += nfeature
            matchsat.saveCNF(cnf, filename)
            return result
    return 'There is no justification based on these axioms'
```

Listing 4.9: Python implementation of BASIS-SEARCH

The function `basisSearch` searches over all the possible non-empty subsets of the given axioms, whether they are a basis for a justification for the feature, using for this check the function `isJustification`. Once it has found a set of axioms that correspond to a normative basis for a justification, it recovers the names in a list (`result`), saves their encoding together with the well-definedness constraint for the mechanism (in `cnf`) and prints the set of names of the axioms. If it searches over all the subsets and none of them is a normative basis for a justification, it returns a string stating it.

Observe that the implementation of the function `subsets` is relevant for the function

`basisSearch` because the basis it returns depends on the order of the subsets it searches over; in the case that there is more than one. Furthermore, recall that every non-trivial superset of a normative basis is also a normative basis (Observation 3.5). In general it could be the case that we are interested in having minimal normative bases, if that's the case, it is important that the implementation of the function `subsets` allows that. Our implementation of `subsets` doesn't return a superset before any of its subsets. Hence, the normative basis that it returns is always minimal. Other implementations of the function `subsets` are possible.

The part of the code that we just explained corresponds to the procedure described in the algorithm BASIS-SEARCH, i.e., to the search of a normative basis. We now illustrate with an example what an execution would look like.

Example 4.1. Recall Example 3.3.

In the Python script we implemented a function `getProfileThree` that, when the problem dimension is $n = 3$, given the agents' preferences as tuples, it retrieves the encoding of a profile as an integer. The profile from Example 3.3 can be represented as in the matrix in Equation 4.5. Recall the encoding of the agents (Section 4.3.1), they are encoded by the integers from 0 to $n - 1$, so in the matrix the left column represents the preferences of the left agents over the right agents and similarly, the right column represents the preferences of the right agents over the left agents. For instance, agent ℓ_1, prefers agent r_1 over r_2 over r_3.

$$\boldsymbol{p} = \begin{pmatrix} 012 & 201 \\ 021 & 012 \\ 210 & 120 \end{pmatrix} \tag{4.5}$$

So, the encoding of profile $\boldsymbol{p}$ is the number 24378. We can corroborate this thanks to the function `prefers` that takes a profile (its integer representation), an index (for the agent) and the type of agent (0 for left, 1 for right), and returns the preference order of the agent under the profile.

Then, we need to generate the list of axioms that will correspond to the corpus $\mathbb{A}$. In Example 3.3, $\mathbb{A}$ is any superset of $\{\mathsf{LSP}, \mathsf{TOPSTA}\}$. So let's take the corpus as $\mathbb{A} = \{\mathsf{LSP}, \mathsf{TOPSTA}, \mathsf{GI}\}$. The code to generate the list `axioms` that we will use as input is the following.

```
axioms = [['TOPSTA',matchsat.cnfTopStable()],
          ['GI',cnfGroupIndifferent()],
          ['LSP', matchsat.cnfLeftStrategyProof()]]
```

Then, the feature φ from Example 3.3 is $\neg\,(1 \leftrightarrows 3)$. The negation $\neg\varphi$ is $(1 \rightleftarrows 3)$, the

encoding of which is the variable $x_{\boldsymbol{p},1,3}$. We generate this encoding automatically and save it in a varibale `nfeature` in the script by executing the following.

```
nfeature = [matchsat.posLiteral(24378,0,2)]
```

Now we can execute the function `basisSearch` with input `axioms`, `nfeature` and `filename`, with the file name being, for example, `'example33.dimacs'`. To do this, we execute `basisSearch(axioms,nfeature,'example33.dimacs')`.

Finally, the output of the execution is `['TOPSTA','LSP']`, which means that the set $\mathcal{A} = \{\mathsf{TOPSTA}, \mathsf{LSP}\}$ is a basis for a justification for the feature φ.

Furthermore, in the directory where we have the Python script, the CNF formula encoding the set $\mathcal{A}$ was saved under the name `example33.dimacs`. For the next step towards obtaining the complete justification, we will make use of this file. ⊣

If a normative basis is found by the function `basisSearch`, there are still some steps remaining to obtain a proper justification. Ultimately, in practice we are more interested in the explanation since that is what we can present the users with. The set of clauses corresponding to $\Delta \cup \{\alpha\}$ in the JUST-SEARCH algorithm (Algorithm 3), is exactly the set of clauses (`cnf`) that were generated by the function `basisSearch` and stored in the file which name we provided as part of the input for it. So the next step is the extraction of an MUS from this set.

For the extraction of an MUS from a set of unsatisfiable clauses, we use the tool PICOMUS, which is a SAT solver that generates an MUS using the library PICOSAT. We do this separately from the Python script we used fro the previous steps.

Example 4.2. In this example we will explain how we extracted an MUS for the set of unsatisfiable clauses saved in a file with name `'example33.dimacs'` generated by the function `basisSearch` in Example 4.1.

We run the following command assuming that PICOSAT and PICOMUS are installed in a directory named `solvers` and that we want the MUS (in DIMACS format) to be saved under the name `example33MUS.dimacs`.

```
~/solvers/picomus example33.dimacs example33MUS.dimacs
```

The output of the solver is the following.

```
s UNSATISFIABLE
c [picomus] computed MUS of size 3 out of 2146177 (0%)
v 365676
v 2123907
v 2146177
v 0
```

This means that the given set of clauses was unsatisfiable and it was able to reduce it from a set with 2146177 clauses to an MUS with only 3 clauses. This is now much

more manageable and it is very likely that we can interpret it and transform it into a human-readable explanation.

The MUS is saved in a text file under the name `example33MUS.dimacs` and it contains the following information.

```
p cnf 419904 3
-219405 -219422 0
219422 0
219405 0
```

The text file follows the DIMACS format. There is one clause in each line and the line breaks are indicated by a 0. The two numbers in the first line (`p cnf 419904 3`) indicate the number of variables and the number of clauses. Even though in this CNF there only appear two different variables, there were 419904 variables in the original CNF. This file format is automatically generated by the function `saveCNF` used in `basisSearch` (Listing 4.9).

By Proposition 4.2, this MUS is equivalent to an explanation for the feature $\neg\,(1 \rightleftarrows 3)$. But we still have to interpret the variables to present it as a logical and human-readable argument. ⊣

Now, we explain how the interpretation of an MUS in DIMACS format can be done. For this we use an implementation of an interpretation function in the original Python script (Endriss, 2019). The function is called `interpretVariable` and it takes as input an integer k and returns the number of profile p, the index of the left agent i, the index of the right agent j and if the literal is a variable or a negation of a variable, for the variable $x_{p,i,j}$ for which k is an encoding. This function can help with reconstructing the instances of the axioms for which the clauses in the MUS are the encodings.

Example 4.3. An example of the execution of the function `interpretVariable`[9] for the variables of the first clause of the MUS in the previous example (Example 4.2) is as follows.

```
>>> interpretVariable(-219405)
-> in profile number 24378 do not match 0/2
-> where profile 24378 = (0>1>2 0>2>1 2>1>0|2>0>1 0>1>2 1>2>0)

>>> interpretVariable(-219422)
-> in profile number 24380 do not match 0/1
-> where profile 24380 = (1>0>2 0>2>1 2>1>0|2>0>1 0>1>2 1>2>0)
```

[9]This function was re implemented in the file `just.py`. A version of it exists in the file `matchsat.py` but it only works for positive literals. In `just.py` we extended it for any kind of literals, but only for matching problem sizes of 2 and 3.

By interpreting the rest of the variables, it is not hard to notice that the first clause corresponds to an instance of left-strategyproofness, because if either of those situations happened, then the left agent 0 could manipulate the situation in profile 24378 by reporting the untruthful preference that results in profile 24380. Furthermore, the second clause of the MUS corresponds to an instance of top-stability and the last one is the one corresponding to the feature, so we will remove it to construct the explanation.

Using this information, we can construct the following human-readable explanation.

If left agent 0 were to be matched with right agent 2 under profile 24378, then she would be able to manipulate by reporting preference order $1 \succ 0 \succ 2$ instead because under profile 24380, left agent 0 and right agent 1 rank each other as their top alternatives. Thus, by top-stability, they should be matched to each other and right agent 1 is a better option for left agent 0 than right agent 2 under her true preference. Hence, grounded in the principles of left-strategyproofness and top-stability, and under profile 24378, the only option is that the pair $(0, 2)$ is not included in the outcome. ⊣

Note that given the MUS in DIMACS format, there is no general way to translate it back into axioms. This step requires some dexterity. For some clauses, it is easily verifiable once we interpret back the variables, but it can be that for some other clauses it's not clear to which axiom they belonged in the first place. However, we could, for example, use the help of the computer to check, for each clause, of which axiom it's an instance. But in general, this part is not done fully automatically and hence, we say that the whole process of producing a justification for a feature is semi-automatic.

4.4 Results

In this section we will take a closer look at the program, how well it performs and what can we achieve by using it. We show some basic statistics of its performance and some examples of usage.

The Python script was run in a regular computer with an 1.8GHz dual-core Intel Core i5 processor and 8 GB of memory. We now give an overview of the time[10] (in seconds) that it took for this computer to generate the encodings of the axioms with matching problem sizes of 2 and 3 and the number of clauses each encoding consisted of. This information is summarized in Table 4.3.

Notice that both, the size of the encodings and the time it takes to compute them grow exponentially. For a matching problem of size $n = 4$ it is not feasible for a regular

[10]Note that the computation time can differ per execution. The times shown in Table 4.3 are an example of one specific execution. However, the encoding sizes never change.

AXIOM	SIZE ($n = 2$)	TIME ($n = 2$)	SIZE ($n = 3$)	TIME ($n = 3$)
Stability	16	0.00078	419904	10.17189
Top-stability	16	0.00028	46656	1.65555
Left-strategyproofness	32	0.00056	2099520	26.99045
Right-strategyproofness	32	0.00077	2099520	32.73899
Strategyproofness	64	0.00135	4199040	64.51469
Group-indifference	64	0.00012	419904	0.82161
Peer-indifference	128	0.00240	3359232	26.86091
Left top-rewarding	16	0.00012	46656	0.62262
Right top-rewarding	16	0.00011	46656	0.51550
Top-rewarding	32	0.00024	93312	0.99002
Left-swap-stability	8	0.00067	209952	6.76091
Right-swap-stability	8	0.00042	209952	7.19294
Swap-stability	16	0.00070	419904	12.81307
No-bottoms	16	0.00027	46656	1.48734
TOTAL	464	0.00897	13716864	196.14219
AVERAGE	33	0.00060	979776	12.01251

Table 4.3: Number of clauses and time in seconds to compute the axiom encodings for matching problem sizes of 2 and 3.

computer to generate the encoding of any of the axioms in reasonable time. For example, when $n = 4$, there are $(4!)^8$ profiles, which is 110075314176 (more than 110 billion profiles!). So, it is reasonable to assume that some axioms would have around that, but very likely more than that, number of clauses. Hence, we were only able to run the program for matching problem sizes of 3 or less.

As a first example of usage of the program, we computed all the normative bases for a justification problem of size 3. Recall Example 3.3. We showed, together with Example 3.3 and Example 3.4, that the feature $\varphi = \neg(1 \rightleftarrows 3)$ could be justified with two different normative basis. It is then natural to ask whether there are more combinations of axioms that could justify φ but to check this manually is a hard task. However, we can use the program that we implemented to search for all the possible normative bases within a set of axioms. It can be done with the function `allBasesSearch`. Recall that the search space when the corpus has size k is of size $2^k - 1$. So, if we include 14 axioms, the search space would be of size 16384. This means that the function `isJustification` would be executed that many times. Hence, for reasons of computational power, we chose a subset of size 7 of these axioms as corpus. The corpus we chose included: stability, top-stability, left strategyproofness, right strategyproofness,[11] group-indifference, peer-

[11]Notice that by including both left and right strategyproofness, we are basically checking for strate-

indifference and no-bottoms. Then, $\mathbb{A} = \{\mathsf{STA}, \mathsf{TOPSTA}, \mathsf{LSP}, \mathsf{RSP}, \mathsf{GI}, \mathsf{PI}, \mathsf{NBOT}\}$.

We ran the function `allBasesSearch` for $\mathbb{A}$, profile $\boldsymbol{p}$ as in Example 3.3, and feature $\neg(1 \rightleftarrows 3)$. The summary of all the bases returned by the program is shown in Table 4.4. Since the corpus consists of 7 axioms, the search space is of size 127. That is, the function checked 127 times if a set formed a normative basis for the feature. It took for the computer to execute the function approximately 10 minutes.

NUMBER	BASIS
1	{STA}
2	{STA, TOPSTA}
3	{STA, LSP}
4	{TOPSTA, LSP}
5	{STA, TOPSTA, LSP}
6	{STA, RSP}
7	{TOPSTA, RSP}
8	{STA, TOPSTA, RSP}
9	{STA, PI}
10	{STA, TOPSTA, PI}
11	{STA, LSP, PI}
12	{TOPSTA, LSP, PI}
13	{STA, TOPSTA, LSP, PI}
14	{STA, RSP, PI}
15	{TOPSTA, RSP, PI}
16	{STA, TOPSTA, RSP, PI}
17	{NBOT}
18	{LSP, NBOT}
19	{RSP, NBOT}
20	{LSP, RSP, NBOT}[12]
21	{PI, NBOT}
22	{LSP, PI, NBOT}
23	{RSP, PI, NBOT}

Table 4.4: All normative bases for the justification problem from Example 3.3

The bases returned in places 4 and 17 of the table correspond to Example 3.3 and Example 3.4, respectively. Indeed, there are many more sets of axioms that can form a normative basis for a justification for φ.

Observe that not all of the computed bases are minimal (Definition 3.3). For instance, every set $\mathcal{A}^N$ in the list such that $\{\mathsf{STA}\} \subset \mathcal{A}^N$ is not a minimal basis since $\{\mathsf{STA}\}$ forms

gyproofnes too. If a basis contains both one-sided versions of strategyproofness, we can replace it by strategyproofness.

[12] This normative basis is actually equal to the set $\{\mathsf{SP}, \mathsf{NBOT}\}$.

a basis. It is nevertheless not necessarily the case that when extracting an explanation for the feature grounded in one of the non-minimal normative bases, there are instances of all the axioms in the explanation. However, a couple of disjoint normative bases were obtained, which implies that we could obtain a number of different explanations for the same feature. This can be useful in a situation where we want to justify the feature for different agents. Each agent may find different principles appealing, so we could present them with the justification that is closer to the axioms they are interested in.

In the next example, we illustrate how often a feature can be justified by at least one set of axioms from the ones we defined. We ran 100 times the following process for matching problems of size 3. First, randomly pick a profile. Then, compute the outcome with the deferred acceptance algorithm (Definition 2.1). Finally, search for a normative basis for each of the pairs in the computed outcome. For each of the 100 runs we registered: (i) how many of the features were justifiable; (ii) with which axioms; and (iii) how long the search for a normative basis took. Again, we had to restrict the set of axioms to a small set for the computations. We used the same corpus in every run of the process: $\{\mathsf{LSP}, \mathsf{RTR}, \mathsf{NBOT}, \mathsf{TOPSTA}, \mathsf{STA}, \mathsf{LSS}, \mathsf{PI}\}$. The choice of this set was arbitrary. Notice that the deferred acceptance algorithm does not satisfy the axiom of right top-rewarding[13] (Definition 2.10). Thus, no normative basis should include it.

In Figure 4.1 we show a graph that summarizes, for each profile, the number of features for which a normative basis was found. On the x axis, we show the encoding of each profile as an integer. The profiles are shown in the same order they were randomly drawn throughout the 100 runs. For each profile, we searched for a basis for three features: the ones corresponding to each pair of the matching outcome computed by the deferred acceptance algorithm. For example, if under a certain profile $\boldsymbol{p}$ the algorithm computes the matching $\{(\ell_1, r_2), (\ell_2, r_3), (\ell_3, r_1)\}$, the features for which we searched for a justification, for profile $\boldsymbol{p}$, were: $(1 \rightleftarrows 2), (2 \rightleftarrows 3)$ and $(3 \rightleftarrows 1)$. On the y axis we show for how many of these features a normative basis was found, for each profile. Observe that for almost every profile, the three pairs forming the matching outcome could be justified with at least one normative basis. Furthermore, there was no profile for which none of the pairs of the outcome could be justified.

In Figure 4.2 we show the time in seconds that the basis search took for every feature. On the x axis of the graph we have the integers encoding the features. It can be observed how for most of the features, it took less than 100 seconds to compute a normative basis

[13]Consider the profile where each left agent ℓ_i ranks r_i at her top preference and each r_i ranks ℓ_i at her bottom preference, for $i \in \{1, 2, 3\}$. The deferred acceptance algorithm returns the matching $\{(\ell_1, r_1), (\ell_2, r_2), (\ell_3, r_3)\}$ in which no right agent is paired with her top option.

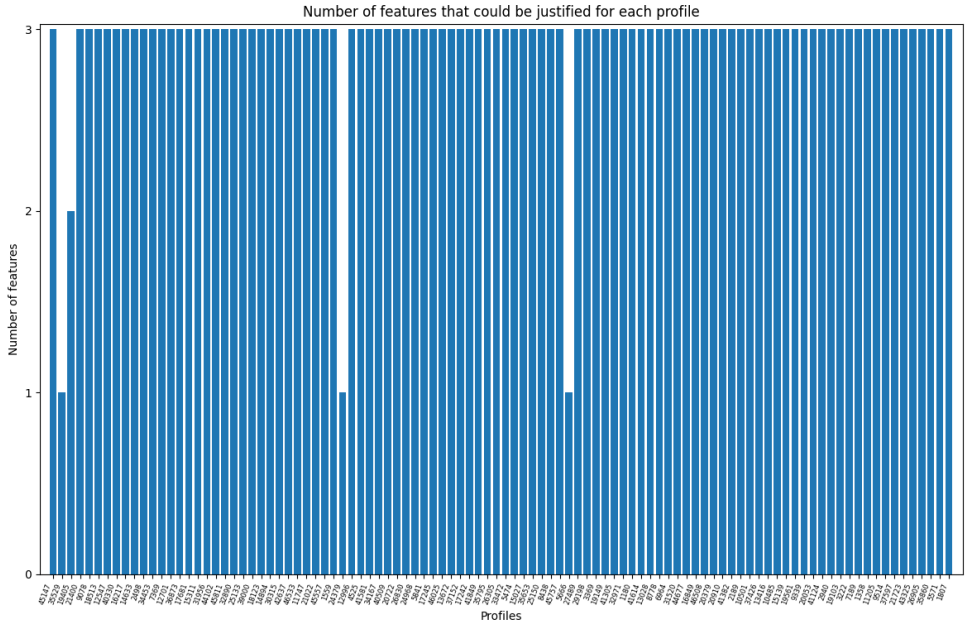

Figure 4.1: Number of features for which a normative basis was found per profile.

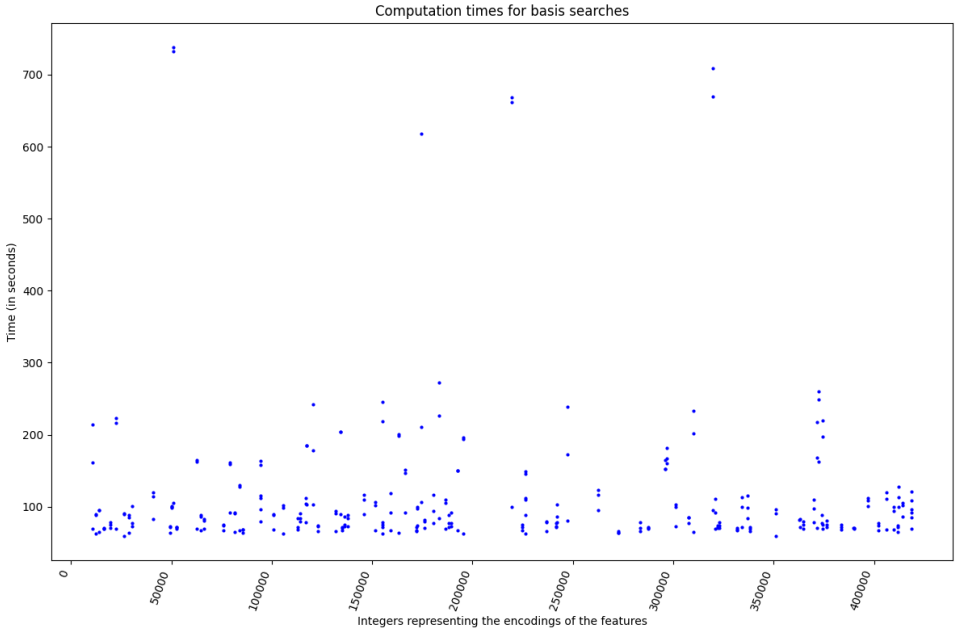

Figure 4.2: Time (in seconds) that it took for for the program to find (or say that there were none) a normative basis for each feature.

for a justification. Then, there is a second group for which it took between 100 and 300 seconds. And there are a few features for which it took much longer (around 11 minutes). Those features are exactly the ones for which no basis could be found among the axioms in the corpus. That is because for those features, the program had to check all the non-empty subsets of the corpus. It took around 9.5 hours for all the runs to execute on a regular computer.

In Listing 4.10 we show part of the output of the experiments run. The first number in each line is the profile under which the deferred algorithm computed the matching. Then there is a list with three elements, each of which consists of a number, encoding a feature, and the basis found for a justification for this feature, or the indication that no basis was found. For instance, in the first run (first line of Listing 4.10) the randomly chosen profile was the one encoded as the number 45147, which corresponds to the following:

$$\begin{array}{ll} \ell_1 : r_2 \succ r_3 \succ r_1 & \quad r_1 : \ell_3 \succ \ell_2 \succ \ell_1 \\ \ell_2 : r_1 \succ r_2 \succ r_3 & \quad r_2 : \ell_3 \succ \ell_1 \succ \ell_0 \\ \ell_3 : r_1 \succ r_2 \succ r_3 & \quad r_3 : \ell_3 \succ \ell_2 \succ \ell_1 \end{array}$$

The deferred acceptance algorithm returns the following matching for this profile.

$$M = \{(\ell_1, r_2), (\ell_2, r_3), (\ell_3, r_1)\}$$

The features $(1 \rightleftarrows 2)$, $(2 \rightleftarrows 3)$ and $(3 \rightleftarrows 1)$ are encoded as the numbers 406325, 406329 and 406330, respectively. As shown in the first line of the output (Listing 4.10), normative bases to justify these features are $\{\mathsf{STA}\}$, $\{\mathsf{STA}\}$ and $\{\mathsf{TOPSTA}\}$, respectively.

```
45147 [[[406325], ['STA']], [[406329], ['STA']],
        [[406330], ['TOPSTA']]]

35529 [[[319764], ['No basis']], [[319766], ['STA']],
        [[319768], ['No basis']]]

19405 [[[174648], ['STA']], [[174649], ['LSP', 'TOPSTA', 'LSS']],
        [[174653], ['No basis']]]

21400 [[[192603], ['LSP', 'TOPSTA']], [[192605], ['LSS']],
        [[192607], ['LSS']]]

9078 [[[81703], ['TOPSTA']], [[81708], ['STA']],
        [[81710], ['STA']]]

18513 [[[166619], ['STA']], [[166621], ['LSS']], [[166626], ['LSS']]]
```

```
12547 [[[112926], ['TOPSTA']], [[112928], ['TOPSTA']],
        [[112930], ['TOPSTA']]]

40330 [[[362971], ['LSP', 'TOPSTA']], [[362975], ['LSP', 'TOPSTA']],
        [[362979], ['TOPSTA']]]

16217 [[[145956], ['STA']], [[145958], ['STA']], [[145960], ['STA']]]

14633 [[[131700], ['TOPSTA']], [[131702], ['STA']],
        [[131704], ['STA']]]
```

Listing 4.10: Normative bases found for the features corresponding to the 10 first profiles

The bases found throughout the computations of the deferred acceptance algorithm under different profiles were quite diverse. Although, as it was expected, the axiom right top-rewarding didn't show up in any basis. This example shows how, for outcomes computed by a common matching mechanism such as the deferred acceptance algorithm, we can almost always find a normative basis to justify each of the pairs forming them. Furthermore, it shows that different parts of an outcome can indeed be justified by different normative bases.

A single run of this process can be useful in a situation where a central authority computes a matching using a mechanism, say like in the example, the deferred acceptance algorithm. Recall Example 3.1, where a matching is to be computed for a mentorship program. In that case, the central authority may be interested in accompanying the communication of the pairing results with an explanation for each agent of why it is the case that they got their assignation.

A final observation on the results obtained with the last example is that once we obtained a normative basis that can justify each of the pairs belonging to the outcome, one could apply the results from Section 3.3 and bring together those justifications to get a justification of the whole outcome or a bigger part of it.

These examples illustrate some situations where the program that we implemented can be useful. Furthermore, they give an idea of the performance of the program.

4.5 Discussion

In this section we present an analysis of the approach and the implementation of the justification search. We discuss advantages, disadvantages and give some pointers on how it could be improved.

SAT solvers are powerful tools, we saw how to transform the problem of normative bases search into a propositional satisfiability problem and with the help of a SAT solver, check the satisfiability of relatively big formulas in a matter of seconds. However, this

was only feasible, for a regular computer, for matching problems of at most size 3. This is the greatest limitation of our approach, since in many real-world situations, there would be more than three agents in each group. On the other hand, the reliability of the approach could also be questioned. In Section 4.2, we discussed how a procedure that is executed by a computer can be subject to errors and could not be considered formally valid. Although the bases search is left to the computer, as we showed, most of the times we are able to obtain an MUS and manually transform it into a human-readable argument. This argument then becomes a reliable proof that the normative basis found by the program can indeed be part of a justification. Nevertheless, we can say that the program can be reliable enough since the encoding is done in such a way that it is clear that the formulas generated by the program are indeed encoding the principles that they are supposed to.

Regarding the computational power limitations, there are a few things that can be improved in the implementation that could make it significantly faster. We could make use of some kind of heuristics. For instance, the search space in the functions `basisSearch` and `allBasesSearch` was the whole power set of the corpus (without the empty set). That is, for every non-empty subset of axioms of the corpus, the functions[14] executed the sub-routine `isJustification`, that verified if a set of axioms could form a normative basis for a justification. But by the definition of a justification (Definition 3.2) we know that a trivial set of axioms cannot constitute a normative basis. We also know that some impossibilities (Section 2.3) hold for certain dimensions n. Hence, we could remove all the sets that correspond to these impossibilities, together with all its supersets from the search space to make it smaller.

Although the approach that we presented proved to be powerful and useful in various situations, it has some limitations and its reliability can be questioned. However, we presented some ideas on how these problems could be minimized.

4.6 Summary

In this chapter we presented and analyzed an approach to automate the justification search using SAT solvers. We gave an overview of the general method and how it has been used in the field of computational social choice. Then, we explained how to apply the basic ideas of this technique to our problem of automatically generating a justification. To do this, we first laid out the process of encoding a justification problem as a propositional

[14]The function `allBasesSearch` did check for every non-empty subsets, while `basisSearch` only did it until it found one. Only in the case where there is no normative basis, it would search over all the non-empty subsets.

formula. Then we formalized the process in the algorithm JUST-SEARCH and prove its correctness. We presented our implementation of the basis search in a Python script and explained how to semi-automatically extract a justification.

Moreover, we showed some examples of how the program could be used. Firstly, an example where we obtained several different normative basis for a justification problem. Secondly, an example that shows that we can indeed justify most of the parts of a matching outcome, computed by the deferred acceptance algorithm, even when taking a restricted corpus of axioms. Finally, we discussed the usefulness and limits of this approach and we hinted at some ways of improving it.

5 | Conclusion

We have defined a formal model for the justification of partial matching outcomes. Furthermore, we have presented an approach to automate the search for such justifications using SAT solvers. We saw that it can often be the case that agents are not happy with the pair assigned to them by a matching mechanism. Also, they might not be interested or able to learn all the theory that could clarify why, if we respect certain normative principles, there is no other option for them to be matched with. In these situations justifications are useful. The idea of providing the agents involved in a collective decision with a justification of the outcome is fairly recent. It has been mostly developed in the context of voting, but in fact, it gives a whole new twist to social choice theory. After all, if we are interested in studying collective decisions, we should also try to make it easier for the agents to come to this decisions. Hence the interest in bringing these ideas to another setting: matching. The main difference with the voting setting is that in matching it is not so interesting in practice to justify the whole outcome. Matching outcomes consist of various pairs that can be seen as "local outcomes". Thus, we were interested in providing justifications for these smaller entities.

In the first part of the thesis, we successfully defined a model for the justification of partial outcomes with the help of the feature language. Moreover, we proved that the justification model behaves as one would expect it to in relation to the kinds of feature formulas that are being justified. We also showed under which conditions it is possible to construct a justification from others. This becomes useful, for example, when we want to justify a partial outcome consisting of various pairs and we have a justification for each of them, in which case we wouldn't need to search for a justification from scratch. Another difference with the justification model for voting is that we didn't require minimality for the explanation in a justification. We discussed this property both for normative bases and for explanations. In the case of normative bases, minimality does not always make the justification more understandable. For explanations, minimality is usually a desirable property but requiring it in the definition imposed some limits for putting justifications together. Nevertheless, we described a process to make an explanation minimal and we

showed that it is always possible to do so. This completed the theoretical analysis of the model.

The second part of the thesis was concerned with the automation of the process of searching for a justification. We defined an algorithm to search for a justification and proved its correctness. The idea behind the algorithm was to transform a justification problem into a propositional satisfiability problem to search for a normative basis and generate an explanation using a minimal unsatisfiable subset. The encoding of the axioms and the matching problem were done as in the classic technique to search (and prove) impossibility results. We presented the implementation of a program that automatically searched for a normative basis for a justification and showed how to extract an explanation using SAT solvers. Our approach for the extraction of an explanation only worked when the negation of the feature could be encoded in a propositional formula with a single clause. However, we discussed how to circumvent this issue. We could find an equivalent feature formula and search for a justification for that one instead, since a justification for a feature formula also justifies all its logically equivalent formulas. We showed examples of interesting ways in which the program could be used. First, we obtained all the possible normative bases for the justification for a feature that we knew could be justified by at least two different bases. This was interesting because we were only able to find two of them by hand and in only 10 minutes, the computer could find all 23 of them. Second, we ran a small experiment that showed that very often we can find a justification for every pair in a matching outcome computed by the deferred acceptance algorithm, grounded in a limited corpus of axioms. In sum, we managed to define a relevant model for justifications for partial one-to-one matching outcomes and we took some steps towards the automation of the justification search. The program we implemented served its purpose but as we discussed, it only worked for small matching instances.

There are still some directions to explore for the further development of this topic. First, we only worked with one-to-one matchings with two-sided preferences and although this model already has many applications, it is the simplest one and there are many others that have important real-world applications. Then, a first direction to explore would be the generalization of the justification model for other matching problems. A first natural generalization of one-to-one matchings with two-sided preferences is the hospital/residents problem. In fact the latter is a one-to-many generalization of the former. For any generalization, the feature language must be adapted in such a way that it captures the corresponding notion of a well-defined matching. In our case, a matching was a pairing such that every left agent was matched to exactly one right agent and vice versa. In the case of the hospital/residents problem, it would be a pairing such that every

left agent is matched to at most one right agent and every right agent is matched to at most a number of left agents equal to its capacity. Once a justification model is defined for other matching problems, we will be able to use it in practice for real-world issues, such as the school choice problem in Amsterdam (de Haan, 2017). Every year, when students apply for secondary schools, since there are schools more popular than others and the spots are limited on every school, some students are not accepted in their top choices. The normative properties and the performance in practice of different algorithms have been analyzed (de Haan et al., 2015; Mennle and Seuken, 2014) and the system has been adapted several times, but every year some parents bring their cases to the courts arguing that their children have not been treated fairly. This is an example in which we would want to provide them with a justification for their assignation to a certain school.

The main drawback of our approach for the automation of the justification search and its implementation is the computational power it needs and that it only works for small instances of a matching problem. This is only expected to get worse when applying it to more general problems. Optimizing the algorithm and the code produced is another line of future work. We already hinted at one way of reducing the search space for a normative basis. We can remove some trivial sets of axioms and all its supersets from the search space. An optimized algorithm for the justification search for voting has been proposed, together with some heuristics (Nardi, 2021).

We didn't analyze the complexity of the justification search problem. A complexity analysis has been done for the justification search problem in the voting context. Boixel and de Haan (2021) proved general results for both the problem of finding and the problem of verifying a justification for an election outcome with two different ways of expressing the axioms. In every case they found that these are computationally hard tasks. As matching instances are bigger in relation to the number of agents than election instances, we expect the complexity of the justification search (and verification) problem to be at least as high. This is another direction for future work, since the application in real-world situations of justification models depends on the feasibility of their computation.

Even though there is still much work to do to further develop this model, we took solid steps towards the definition and automation of a justification model for partial matching outcomes. More generally, we made some progress in the development of models for justifications in different collective decision making settings, which is a growing area in computational social choice.

Bibliography

Abdulkadiroğlu, A. (2013). School choice. In Vulkan, N., Roth, A. E., and Neeman, Z., editors, *Handbook of Market Design*, pages 138–169. Oxford University Press.

Abdulkadiroğlu, A. and Sönmez, T. (2003). School choice: A mechanism design approach. *American Economic Review*, 93(3):729–747.

Arora, S. and Barak, B. (2009). *Computational complexity: A modern approach*. Cambridge University Press.

Arrow, K. (1951). *Social choice and individual values*. New Haven, CT: Cowles Foundation.

Biere, A. (2008). PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):75–97.

Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors (2009). *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications, v. 185. IOS Press, Amsterdam, The Netherlands.

Boixel, A. and de Haan, R. (2021). On the complexity of finding justifications for collective decisions. *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 35(6):5194–5201.

Boixel, A. and Endriss, U. (2020). Automated Justification of Collective Decisions via Constraint Solving. *Proceedings of the 19th International Conference on Autonomus Agents and Multiagent Systems (AAMAS-2020)*.

Brandl, F., Brandt, F., Geist, C., and Hofbauer, J. (2015). Strategic abstention based on preference extensions: Positive results and computer-generated impossibilities. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-2015)*.

Cailloux, O. and Endriss, U. (2016). Arguing about voting rules. *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2016)*.

de Haan, M., Gautier, P. A., Oosterbeek, H., and Van der Klaauw, B. (2015). The performance of school assignment mechanisms in practice. In *IZA Discussion Papers 9118*. Institute of Labor Economics (IZA).

de Haan, R. (2017). Why a Dutch court stopped high school students from swapping schools. Medium.com. Available at https://tinyurl.com/3tv5c88b. Site visited on 7 January 2022.

Endriss, U. (2019). Software and data for "Analysis of one-to-one matching mechanisms via SAT solving: Impossibilities for universal axioms". Zenodo. https://doi.org/10.5281/zenodo.3547826.

Endriss, U. (2020). Analysis of one-to-one matching mechanisms via SAT solving: Impossibilities for universal axioms. In *Proceedings of the 34th AAI Conference on Artificial Intelligence (AAAI-2020)*.

Gale, D. and Shapley, L. S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15.

Geist, C. and Endriss, U. (2011). Automated search for impossibility theorems in social choice theory: Ranking sets of objects. *Journal of Artificial Intelligence Research*, 40:143–174.

Geist, C. and Peters, D. (2017). Computer-Aided Methods for Social Choice Theory. In Endriss, U., editor, *Trends in Computational Social Choice*, chapter 13, pages 249–267. AI Access.

Gibbard, A. (1973). Manipulation of voting schemes: A general result. *Econometrica: Journal of the Econometric Society*, 41:587–601.

Klaus, B., Manlove, D. F., and Rossi, F. (2016). *Handbook of Computational Social Choice*, chapter 14, pages 333–355. Cambridge University Press.

Manlove, D. (2013). *Algorithmics of matching under preferences*. World Scientific.

Masarani, F. and Gokturk, S. S. (1989). On the existence of fair matching algorithms. *Theory and Decision*, 26(3):305–322.

Mennle, T. and Seuken, S. (2014). Trade-offs in school choice: Comparing deferred acceptance, the naive and the classic Boston mechanism. *arXiv preprint arXiv:1406.3327*.

Nardi, O. (2021). A graph-based algorithm for the automated justification of collective decisions. Master's thesis, University of Amsterdam.

Okada, N., Todo, T., and Yokoo, M. (2019). SAT-Based automated mechanism design for false-name-proof facility location. In Baldoni, M., Dastani, M., Liao, B., Sakurai, Y., and Zalila-Wenkstern, R., editors, *In Proceedings of the 22nd International Conference on Principles and Practice of Multi-Agent Systems (PRIMA-2019)*, volume 11873 of *Lecture Notes in Computer Science*, pages 321–337. Springer.

Procaccia, A. D. (2019). Axioms should explain solutions. In *The Future of Economic Design*, pages 195–199. Springer.

Roth, A. E. (1982). The economics of matching: Stability and incentives. *Mathematics of Operations Research*, 7(4):617–628.

Roth, A. E. (1984). The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, 92(6):991–1016.

Roth, A. E., Sönmez, T., and Ünver, M. U. (2004). Kidney exchange. *The Quarterly Journal of Economics*, 119(2):457–488.

Roth, A. E. and Sotomayor, M. (1992). *Two-sided matching*, volume 1 of *Handbook of Game Theory with Economic Applications*, chapter 16, pages 485–541. Elsevier.

Satterthwaite, M. A. (1975). Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217.

Tang, P. and Lin, F. (2009). Computer-aided proofs of Arrow's and other impossibility theorems. *Artificial Intelligence*, 173(11):1041–1053.

Tang, P. and Lin, F. (2011). Discovering theorems in game theory: Two-person games with unique pure Nash equilibrium payoffs. *Artificial Intelligence*, 175(14):2010–2020.